A SEMANTIC MODELLING APPROACH TO KNOWLEDGE BASED STATISTICAL SOFTWARE

KEVIN WILLIAM LAWSON

Doctor of Philosophy

THE UNIVERSITY OF ASTON

February 1989

The University of Aston

A SEMANTIC MODELLING APPROACH TO KNOWLEDGE BASED STATISTICAL SOFTWARE

Kevin William Lawson

Doctor of Philosophy

1989

Summary

The topic of the thesis is the development of knowledge based statistical software. The shortcomings of conventional statistical packages are discussed to illustrate the need to develop software which is able to exhibit a greater degree of statistical expertise, thereby reducing the misuse of statistical methods by those not well versed in the art of statistical analysis. Some of the issues involved in the development of knowledge based software are presented and a review is given of some of the systems that have been developed so far. The majority of these have moved away from conventional architectures by adopting what can be termed an expert systems approach.

The thesis then proposes an approach which is based upon the concept of semantic modelling. By representing some of the semantic meaning of data, it is conceived that a system could examine a request to apply a statistical technique and check if the use of the chosen technique was semantically sound, i.e. will the results obtained be meaningful. Current systems, in contrast, can only perform what can be considered as syntactic checks.

The prototype system that has been implemented to explore the feasibility of such an approach is presented, the system has been designed as an enhanced variant of a conventional style statistical package. This involved developing a semantic data model to represent some of the statistically relevant knowledge about data and identifying sets of requirements that should be met for the application of the statistical techniques to be valid. Those areas of statistics covered in the prototype are measures of association and tests of location.

Keywords : statistical software, knowledge based software, semantic modelling.

Acknowledgements

I would like to thank my supervisor Paul Golder for his much appreciated advice and assistance throughout the stages of this research. My thanks also go to SERC for their financial support.

Contents

Figures

Tables

Chapter 1
Introduction

Although the use of machinery for statistical purposes can be traced back to the production of tables on mechanical calculators, it was not until the arrival of general programming languages, Fortran in particular, that machinery was widely used for statistical analysis.

Single stand-alone programs to perform a specific task can be considered as the first generation of statistical software. These programs were usually written by statisticians for their own personal use and involved a great deal of, often repeated, work. To write a program required a thorough knowledge of the intricacies of the steps constituting the chosen test or technique. A good working knowledge of a general purpose programming language was also called for. Some programs did get published, but due to inconsistencies in data formats there were problems in attempting to run several programs to perform a multi-stage analysis.

Some of the problems of the above approach were to some extent alleviated by the release of libraries of pre-compiled subroutines to perform common well defined tasks. An analysis could then be built up using a piecemeal approach with calls to the appropriate subroutines. This development allowed a user to program at a slightly higher numeric level. For example, being able to call a procedure to perform a matrix inversion rather than having to program the individual steps of a particular algorithm. Good libraries have an advantage in that they provide a set of well designed, tested and consistent subroutines. However, it is still necessary to write the main program to call the subroutines and manipulate the data.

The early 1960's saw the emergence of integrated statistical systems with their own problem orientated command languages. These provided the flexibility of subroutine libraries but allowed users to program at a more statistically orientated level. This obviated the need to know a general programming language, although some early languages were based on Fortran. Also, the batch approach to programming has largely given way to systems able to operate in an interactive manner. This has obvious advantages when working at the exploratory stages of an analysis.

Papers by Chambers (1980) and Nelder (1984) chronicle in more detail the development of statistical computing, discussing computing in general and in particular statistical software.

The most recent research efforts into the further development of statistical software have been concerned with the production of software able to demonstrate some degree of statistical expertise.

In chapter 2, the need to develop software with a greater degree of statistical expertise is examined, illustrating some of the shortcomings of the current generation of software. A review is given of literature which has discussed issues and problems involved in developing knowledge based software and has proposed solutions, with the emphasis of much of the literature being placed upon the application of artificial intelligence techniques. To conclude the chapter, a brief description is given of a number of knowledge based systems that have been under development to date.

An alternative approach to knowledge based statistical software is proposed in chapter 3. By representing more of the semantic meaning of data, it is suggested that conventional style systems could be enhanced and be better able to detect the misuse of statistical methods. The background information to the research proposals is presented, that is the representation of semantic knowledge - in the areas of database management systems and artificial intelligence - and the inclusion of metadata in statistical database management systems.

Chapter 4 provides an overview of the prototype system that has been developed to investigate the viability of the research proposals. The implementation is presented with reference to the main components of the system and the areas of statistics that it supports.

A detailed description of the semantic data model is given in chapter 5. Attention is focused on: the semantic concepts which constitute the knowledge about the data; the symbolic objects that organise and structure the knowledge; the commands and routines to manage and manipulate the objects.

Chapter 6 begins with a brief description of the specific statistical methods that are supported by the system, identifying some of the conditions which should be met for their correct use. The scheme adopted to represent the requirements is illustrated, pinpointing the knowledge in the data model that can be used to validate each of them.

The operation of the system in validating the use of the statistical methods is reported in chapter 7. Included is how the checks are performed and the feedback that is given to the user.

Finally, the conclusions of the research are presented in chapter 8.

Chapter 2

The Development of Knowledge Based Statistical Software

2.1 Conventional Statistical Software

As statistical software has evolved, the knowledge required of someone to use a computer to perform an analysis has gradually diminished. The need to understand or even know about the low level numeric operations of a particular test or technique has all but disappeared, with packages providing high level commands such as *regress (x, y)*. Command languages have become more flexible and natural, moving away from a computer orientated approach. The fixed format numeric codes of the past have given way to a free format English like dialogue. Hand (1985b) noted that there had been a tendency to make statistical software as easy to use as possible, this has mirrored the trend in other areas of applications software, e.g. word processing packages. Indeed, there would seem little point in needlessly making something difficult to use.

Paralleling, or more probably outstripping the developments in statistical software, have been the advances in computer science. Hardware has become more powerful and yet at the same time cheaper. The outcome of which is that computers are now much more readily available.

As a result of the progress outlined above, statistical software has become much more accessible. The ability to apply statistical techniques is no longer the preserve of professional statisticians, researchers in all manner of domains are now able to autonomously analyse their experimental data. A large number of these users can be termed as being statistically naive, that is they may be experts in their own fields of research but have only a limited knowledge and appreciation of statistics.

Statistical software provides the mechanics to perform an analysis and has done well in performing this function (Chambers, 1981), the facilities offered by the larger packages are now reasonably complete. Hand (1984) summed up existing statistical software as containing arithmetic and algebraic expertise, systems know how to compute a test statistic. Given data that is the correct type and of the right shape (e.g. real vectors of the same length), the package can manipulate it to produce the result. However, it can be said that only syntactic checks are performed, current packages are largely unintelligent in that they do not check if the assumptions of the test are met. It is up to the user to know if it is appropriate to apply a particular test. An analysis only produces intelligible results if it is appropriate to the data. With current statistical packages, the user is left to interpret the results obtained and attach meaning to them.

As was mentioned above, many users of statistical software are not themselves experts in the area of statistics, Hooke (1980) remarked that by making statistics more accessible, use had been replaced by overuse and misuse. Unfortunately, incorrect use of statistical software is not reported and users are often unaware of their mistakes and misconceptions. A number of studies into the use of statistical methods in medical journals (Badgley, 1961; Schor & Karten, 1966; Gore *et al.*, 1977; Glantz, 1980; Altman, 1982) have found that approximately 50% of published papers using statistical methods contained inappropriate or incomplete analyses. Errors were found to have occured throughout the stages of applying an analysis, either in the design of an experiment, in calculating results or drawing conclusions.

Nelder (1977) opined that misuse of statistical software was bringing the subject of statistics into disrepute, a sentiment similarly echoed from other quarters. Chambers (1981) advocated that software should do more than merely perform blind computational algorithms and that statisticians had a moral obligation to provide users with better guidance. To that end, there has been a growing interest in developing knowledge based statistical software as a means of providing a greater degree of support to users. It is hoped that some of the misuse can be filtered out, resulting in more correct and appropriate analyses being performed.

2.2 Background to Knowledge Based Statistical Software

An early paper to appear which advocated making statistical software more intelligent was by Nelder (1977). He observed that current software was largely unintelligent as no use was made of the data to check if the assumptions regarding a statistical procedure were satisfied. To illustrate his point, regression analysis was used to discuss checks that could be applied to protect a model from a number of sources of distortion. In addition, it was also noted that to facilitate further checking it would be valuable for a program to require a user to specify information external to the actual data items themselves.

Gale (1986a) remarked that at that time techniques for implementing Nelders ideas lacked the power to achieve the desired objectives. Since then there has been an enormous interest in artificial intelligence (AI) research and the prospects for programs to be able to exhibit some degree of intelligence look brighter.

Chambers (1981) was one of the first to discuss the application of AI techniques to the development of statistical software. In particular, he considered the possibility of expert software being able to perform some of the functions of a consultant statistician. An earlier paper by Jones (1980) had considered the possibility of a computer program being able to act as a statistical consultant. He noted that the attitudes and personalities of both the client and consultant were important factors in the collaboration of the two. To that end, he discussed some negative stereotypes of consultants that a program would need to avoid, as well as good characteristics that ought to be retained. Some negative stereotypes of clients were also highlighted which a program would need to cater for.

The consultant statistican has often been the role model when considering what knowledge based software could be used for (Hand, 1984, 1985a, 1986; Hahn, 1985). By examining some of the functions carried out by a human expert, some potential uses have been identified. Generally speaking, the proposals for systems have encompassed the following five tasks :-

(i) statistical answering and referral services;

(ii) the refinement of research objectives;

(iii) choosing appropriate statistical techniques;

(iv) correctly applying a particular technique;

(v) interpretation of results.

Hahn (1985) described software falling into the first category as requiring only "dumb statistical intelligence". However he saw these type of systems as providing a useful source of information for both statisticians and non-statisticians alike, addressing fairly general and mundane matters. As an answering service, questions regarding statistical techniques such as their use, sources of reference and packages providing an implementation could be dealt with. A computerised statistical index could provide users with information regarding books, articles and available software. By computerising such information, access to it is made much easier - with regards to searching - and the time lag for it to become available is reduced. Hahn recognised that it may be dubious to describe such systems as being intelligent, but felt that they established a precedent for software to provide information regarding statistical methodology, rather than just applying statistical methods. He noted that such systems were feasible and were being implemented in a number of ways.

Software able to assist in refining research objectives and questions was discussed by Hand (1986). He saw such systems as being of benefit during the design stage of an experiment. This could involve ensuring that, at a minimal cost, results are obtained which are both accurate and able to answer the questions of the researcher. This is an important and cost effective exercise, although unfortunately the advice of a statistician is sometimes not sought until after the data has been collected.

The third area where it has been envisaged that knowledge based software could make a contribution is in the choice of an appropriate statistical technique. This could be in the form of a fairly broad system, performing at a very high level and requiring a great deal of expertise, for example conducting a discourse with the user and suggesting that regression analysis would be appropriate for the problem at hand.

Alternatively, if a broad technique had already been decided upon, the choice could be at a much lower level requiring less expertise, an example of this task could be choosing an appropriate bivariate test of location.

The division of expertise between choosing and applying a technique is somewhat blurred. Hand (1985b) noted that if the technique chosen was regression analysis, a considerable amount of expertise and effort would still be required to ensure that the model developed was appropriate. However, once a test of location has been selected the bulk of the work has been done, the effort involved in applying it correctly is considerably less in comparison.

Knowledge based software able to interpret the results of an analysis was briefly covered by Hand (1986) and is an area still to be looked at in any great detail.

As can be seen from the five areas discussed, knowledge based software is seen as an opportunity to broaden the role that software can play in the area of statistics. The type of systems suggested go beyond the scope of existing statistical packages.

Software able to adequately perform all of the functions of a human consultant are a long way off. Jones (1980) felt that only a limited number of problems could be tackled by a computer program, similarly Hahn (1985) thought that there were greater chances for success if effort was concentrated on producing systems for limited specialised tasks. General systems have not been attempted and the tasks which have received the most attention are :-

(i) technique selection - choosing a technique appropriate to the problem;

(ii) technique application - guiding a user to a correct and proper application of an advanced statistical technique, e.g. regression analysis.

Analogies have been drawn between these aspects of statistical consultancy and medical diagnosis, for which the expert system MYCIN (Shortliffe, 1976) has been developed. The most striking similarity which has been identified is one of having to make a choice from a number of alternatives arranged in an ill-structured domain. There are however certain differences which set statistical applications apart. Firstly, statistical systems for data analysis should make use of two sources of information, namely the user and the data. Also, medical systems can assume a user with some knowledge of the domain, which is not true of statistical systems. Indeed, Hand (1985a) noted that when considering the areas for which expert systems had been applied, what was striking were not the similarities but the differences.

In addressing the problem of choice (for regression analysis this would be choosing an appropriate model), a strategy for making this choice has to be educed and represented. In developing statistical expert systems, statisticians have been forced to consider in detail how they go about their own consultancy work. This aspect of consultancy had previously received little attention, but has now become the focus of research effort and has yielded benefits which go beyond the development of computerised consultancy.

Consideration must also be given as to the role that intelligent software should play. If it were possible to produce software able to automatically look at the data, carry out the analysis and output the results, would such systems be desirable? Hand (1984) termed this the oracle approach, both he and Hahn (1985) opined that software assuming such total control was not desirable. One reason for this conclusion is due to the very nature of statistics, which is a discipline that is not used in isolation. Rather, its use is to explore and explain phenomena occuring in a ground domain, that is the discipline it is being applied to. Intelligent statistical software is envisaged to contain statistical knowledge. However, as Hand (1984) noted, "effective statistical work involves a subtle interplay between two types of knowledge, the statistical and that of the domain of study" and preferred to talk of "the notion of expert systems for giving statistical advice". It would not be practical to try and encapsulate into statistical software the required domain knowledge. Since statistics is applied across such a diverse range of disciplines, the work involved to elicit and represent such knowledge would be immense. However, the user of these proposed systems will possess the required domain knowledge and it would seem prudent to make use of this.

Because of the need for domain knowledge, the role that is generally suggested for intelligent statistical software is that of an advisor, to guide the user to an appropriate and sound analysis. When a researcher seeks the advice of a human consultant, a dialogue takes place between the two and both parties are involved in making a decision as to an appropriate course of action. It would seem sensible for intelligent software to assume a similar role.

The stages of an analysis could be considered in a simplified form as in figure 2.1.

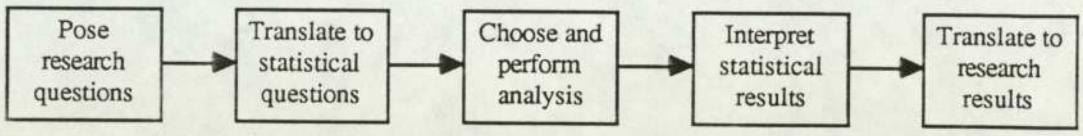
Pose research questions
Translate to statistical questions
Choose and perform analysis
Interpret statistical results
Translate to research results

Figure 2.1 Stages of a Statistical Analysis

Throughout the above stages, a knowledge of the domain of application is required. Initially, the statistician will talk through the problem with the researcher, often in the language of the researcher. The purpose of this is to clarify the objectives of the analysis, of which the client may initially only have a vague notion. The skill of the consultant is to recognise the salient points and to formulate an analysis appropriate to the problem. Conversely, once the results of the analysis have been obtained, the meaning must be conveyed to the researcher in their own language. This task is an important one, as the results of one analysis will often lead to further questions and analyses. Chambers (1981) felt that this ability to bridge the gap between the two fields of knowledge would be a difficult function for software to imitate.

When it comes to using statistical software, non-statisticians may be happy to sit back and be given an answer. However, it is often the case that there is no definitive answer, there are often conflicts which need to be resolved and depending upon their relative importance different solutions will be adopted. For a user that has some statistical expertise, software dictating an answer will not be appreciated. Such a user will wish to compare their own initial ideas with the systems conclusions before deciding on a course of action to embark upon. By involving the user in the decision process, not only will the analysis be better able to answer the required questions, but the user will have a greater understanding of what has been done and why.

The fact that statistical software is used by people with such differing levels of statistical expertise causes further problems in the development of intelligent systems. Although it is convenient to discuss users as being statistically naive or experts, they do in fact fall upon a continuum and not a dichotomy. The needs of these users will differ, indeed the needs of a user will alter over time as expertise is gained. Those with a weak grasp of statistics will require a system to have good explanatory facilities to make clear the unknown, whereas those with a better knowledge will merely want to be reminded of points which may have been forgotten.

2.3 Some Knowledge Based Statistical Software

Initial attempts at introducing statistical expertise as a means of reducing misuse came in the form of interfaces, or front-ends, to existing statistical packages. One in particular was BUMP (Smith *et al.*, 1983), this was written as an interface to the MULTIVARIANCE program which is used for univariate and multivariate analysis of variance, covariance, regression and repeated measures analysis. Although an initial aim was to make access to the package easier for non-statisticians (due to the fact that an analysis is described using an awkward system of numeric codes), the possibilities of preventing misuse and providing pedagogical facilities were also explored. The system operated by questioning the user about their problem and was then able to produce a complete MULTIVARIANCE program. Work on producing interfaces was limited and the majority of research has concentrated on expert systems solutions, although Wolstenholme & Nelder (1986) have recently been working on a front-end to GLIM.

Gale (1985) observed that efforts to apply AI techniques to statistical software were taking two distinct directions :-

(i) providing guidance for those with little training in statistics;

(ii) making software more productive for professionals.

Gale noted that software for experts would require more statistical knowledge to be represented. Since the representation of statistical knowledge for machine use is still in its infancy, he felt that more usable software would be initially developed in the former direction. Some work aimed at the professional has been carried out which has focussed on the study and representation of strategy (Huber, 1986; Oldford & Peters, 1986). However, most systems have concentrated on providing guidance for non-statisticians.

One of the most important of the early expert systems, certainly one which has received the most attention, is REX (Gale & Pregibon,1982; Pregibon & Gale, 1984) which was developed at Bell Laboratories. This is a Regression EXpert and aims to safely guide a user to perform a simple linear regression analysis. The principle it adopts is to consider a user's request to fit a regression model and to check the assumptions underlying the technique. A number of tests are carried out, if a problem is detected the system attempts to find a solution which is acceptable to the user. When there are no problems which remain unresolved, the analysis is complete and the model has been determined. The mechanics of the analysis are performed using the S System (Becker & Chambers, 1984) which has also been developed at Bell Laboratories. As well as providing guidance, Gale and Pregibon also wanted the system to provide instruction and interpretation, to educate users in regression analysis and to be able to explain the meaning of results obtained. They felt that expert systems techniques offered advantages over conventional programming styles in providing these facilities. REX used a combination of production rules and frames to represent its statistical knowledge, a detailed description of which is given in Gale (1985).

Oldford & Peters (1984) have also focussed on multiple linear regression analysis but have tackled development using a bottom up approach. Four major subtasks within linear regression analysis were identified, they decided to concentrate on building prototype systems for those individual tasks before combining them into an overall system. The paper discussed the first such subsystem which addressed the problems involved in detecting and correcting collinearity.

The RX project (Blum, 1982) aimed to tackle the problem of converting a research goal into a statistical goal. It did so by restricting the domain of application to medicine and was able to translate a research question posed in medical terminology into a description of a statistical study capable of answering the question. To achieve this it was necessary for the knowledge base to represent both medical and statistical concepts. The system was used to design studies which could examine the existence of causal relationships in a database of longitudinal medical records. The system was then able to add newly found information to a third portion of the knowledge base concerned with causal relationships.

There were a number of early systems which undertook to explore the difficulties associated with developing software capable of assisting in the selection of an appropriate statistical technique. Notable amongst them were two by Hajek & Ivanek (1982) and Portier & Lai (1983).

Hajek and Ivanek considered the application of AI techniques to software for exploratory data analysis. Their aim was to develop a consultancy system to assist in the use of the GUHA package (a brief description of which can be found in Hajek & Havranek (1978)), which is oriented towards nominal and dichotomous data. A subsystem GQUANT was implemented for the ASSOC procedure, to search for associations in the data. GQUANT had 34 rules, by asking the user up to 9 questions a choice could be made from 6 statistical tests, e.g. chi-square and Fisher exact. The principle adopted was to view data analysis as a search, the goal of which is ill-defined. A fully fledged system, GUHA 80, was planned but never implemented.

A system which attempted to tackle the problem of a user not understanding the meaning of a question was produced by Portier and Lai. The STATistical PATHfinder (STATPATH) was a menu-driven system which identified an appropriate analysis by performing a binary tree search, the tree being represented as production rules. The objective in asking the user questions was to narrow down the field of possibilities. If the user was unable to give a yes or no answer then both lines of questioning were followed up. One problem that exists is that once a number of questions are answered as unknown the avenues to explore increase rapidly. Also, the system is unable to guard against a user mistakingly thinking that they are able to give a definite answer to a question.

Of late, those aspects of statistical work for which expert systems have been developed has widened. Dickson & Talbot (1986) are developing a system for use on a microcomputer to perform data validation functions. The choice of a microcomputer application was to facilitate the use of electronic measuring devices, some data capture procedures were described in Dickson (1984). The aim is for the system, which has been written in BASIC, to monitor data input and to highlight possible errors to the operator. The authors perceive validation as a dynamic process with the system being able to learn from information recorded and the responses of the operator.

A system able to learn a strategy was the aim of an ambitious project that was undertaken at Bell Laboratories. It was envisaged that Student (Gale & Pregibon, 1984; Gale, 1986b) would be able to learn by means of example. When Gale and Pregibon developed REX, a number of regression analyses were performed using the S system. The strategy was constructed by analysing the steps carried out and questioning why certain actions were taken. Gale and Pregibon designed Student to work in the same way, to observe a professional statistician performing an analysis using a statistical package and to ask questions. This approach would allow a professional statistician to develop a knowledge based system for an aspect of data analysis without the need to know about the internal representation. The statistician would first conduct an analysis for a technique that is new to the system. By then adding further examples the strategy can be extended and consolidated. Another perceived advantage of this approach is that it would allow the builder to bias the knowledge and vocabulary towards a specific domain of application. For such a project to succeed enormous problems would have to be overcome and some progress was made. Work on the system has now ceased after four years as Gale and Pregibon could not see it reaching a satisfactory state for at least another year.

Although the majority of knowledge based statistical software that has been under development has been in the form of expert systems, often implemented using production rules, work in other directions has proceeded.

Baines & Clithero (1986) are developing a user-friendly package for the design and analysis of experiments using standard programming methods. They noted that packages in general use were unsafe for an inexpert user because they are predominantly concerned with computational aspects. Although there are often the facilities to perform validity checks, a certain degree of statistical expertise is required of the user to know what checks to perform and how to interpret the results. Baines and Clithero sought to add some consultancy features to aid inexpert users. The structure of the program, which was written in Fortran rather than Lisp or Prolog, was a tree network to represent all possible outcomes. Those outcomes considered as having exceptional combinations of circumstances were not dealt with and the user advised to consult a statistician. The program consisted of a top level overview module to determine which area of experimental design was appropriate for the problem. If successful, control would be passed to a design and analysis module. At that time three such modules were being implemented for simple comparative, factorial and response surface experiments.

The term knowledge enhancement system has been used to describe the KENS program (Hand, 1987), this was felt to be more appropriate than calling it an expert system. Its function is not to guide a user in conducting an analysis but to provide information, in this particular case about nonparametric statistical methods. The objective was to provide a tool to allow a user with some knowledge of the domain to explore the subject and enhance their knowledge. Hand was motivated to develop the system for use in his role as a consultant statistician, to assist him in choosing appropriate tests and to remind him of concepts which may have been forgotten. In designing the system he felt that a production rule architecture would not provide for the flexible type of interaction envisaged. Hand considered that they were well suited for diagnostic problems of choosing from a number of alternatives, but less so when the objective was not as clearly predetermined. The architecture developed was a network representation. Nodes were used to encode portions of text (*frames*) and single words or phrases (*descriptors*). The system consists of three graphs, which can be regarded as semantic networks. The *relationship graph* links descriptors and represents associations between them. Frames are cross referenced in the *reference graph* . The *concept graph* links the frames with the descriptors that define them. Upon invoking KENS the user enters a descriptor which is used to initiate a search for relevant frames. A descriptor can be preceeded by a relation which modifies the search, for example to look for frames which are related to antonyms of the descriptor. If successful, the system will return a list of frames with a ranking as to their likely relevance, the user can either look at a frame or enter a new descriptor. As well as giving textual information, a frame may also give a further list of frames which may be of use. Although not initially intended for the statistically naive, during the course of development KENS has been extended and is now more suitable for such a user. A version of KENS has been made available as a prototype system and is still under development.

2.4 Conclusions of Review

The software described in the previous section does not constitute an exhaustive review of the knowledge based statistical systems that have been produced. It does however illustrate the diversity of the research and development work that has been carried out. Two points have emerged that are of particular interest. An increasing number of aspects of statistical work have been the subject of investigation for the development of knowledge based systems. Secondly, there has been a diversification of the types of systems developed, that is they are not all expert systems with a production rule architecture.

There are as yet no commercially available statistical knowledge based systems, those that have been under development have so far reached the feasibility demonstration stage. This is a reflection of the youth of the field and significant progress has been made towards the production of such a system.

Chapter 3

A Semantic Modelling Approach

3.1 Motivation

Conventional statistical packages which are currently in common use are lacking in that they contain arithmetic and algebraic expertise but little or no statistical expertise. This expertise is left for the user to provide, whose statistical knowledge is often limited, leaving the software open to misuse.

Attempts aimed at producing software able to exhibit some degree of statistical expertise have to a great extent concentrated on using expert systems techniques. This approach seeks to encode a strategy which is able to guide a user to perform a safe and appropriate analysis. A number of methods have been employed to represent the strategy, including production rules and decision trees. The interface to such systems is usually of a conversational type, which endeavours to mimic an interaction with a human expert. In general, expert systems solutions have been developed for relatively narrow topics which require a great depth of knowledge, with regression analysis being a classic example.

The research into expert systems represents a shift away from statistical software built using conventional architectures, which have evolved towards command driven systems encompassing a wide and diverse range of statistical facilities. One of the advantages of such systems, which has influenced development to move in this direction, is that of flexibility. Command driven systems provide a flexible tool to perform data analysis, whereby at any point the user is able to call from a variety of functions. This is particularly useful when performing an exploratory type analysis, where the purpose is to examine and to gain some insight of the data. Use will be made of graphical displays and plots, summary statistics and tests to elucidate relations between samples. In many cases the user will have some initial ideas to explore but will not have a totally predetermined set of commands to enter. The path of such an analysis is determined dynamically with the results of earlier commands having a bearing on its future direction. In such cases, a suitable system would be one which is command driven and enhanced with statistical expertise to offer some protection against possible misuse. In considering expert software for conducting analyses, in contrast to software for choosing appropriate statistical methods, Hand (1985a) considered that sophisticated variants of conventional packages may be the most suitable.

One of the problems with current statistical packages is that when a user issues a command to perform some operation, statistical or numeric, all that the package is able to do is check to see if the command is syntactically correct. That is to see if the correct number of arguments of an appropriate type have been given, for example two numeric vectors of the same length. It would be desirable if packages could be enhanced such that they were also able to check on the semantics of the command. That is to be able to advise on whether or not the test specified is appropriate for the data given, will the result obtained have any sensible meaning. This would involve checking that the conditions regarding the use of the test are not violated. The reason why current statistical packages are unable to offer much assistance to users in this respect is due to the limited amount of knowledge that they have about the data. When data is entered into a statistical package it is typically identified by its type (e.g. numeric, alphanumeric, boolean) and stored in a suitable data structure. To be able to apply a number of semantic checks, and hence advise on the soundness of applying a statistical test, more must be known about the data than its type, some semantic knowledge of what the data represents is required. Some packages, for example SPSS, have represented other information about data such as variable and value labels, but this has been purely for documentation and display purposes. If a system required more semantic information to be entered about the data, possibly in a manner akin to the data definition approach of database management systems, then a model could be constructed to represent some of the semantic meaning of the data. When analysing a request to perform a statistical operation, it would then be possible for a system to report back to the user the result of applying a number of semantic checks which had consulted both the actual data items and also the semantic knowledge about the data.

3.2 Representing Semantic Knowledge

3.2.1 Semantic Modelling

Researchers working in the area of database management systems have also been interested in representing more of the semantic meaning of data. It was proposed that the use of more semantic models would make the database design stage more systematic and that systems based on such models would be able to respond more intelligently to user requests.

The standard database systems (implemented with relational, network or hierarchical models) are not totally devoid of semantic information but have only a very limited understanding of the meaning of the data. The objective has therefore been to extend the knowledge represented in these existing data models, to add on an extra layer. This task of representing meaning has been termed *semantic modelling*. Codd (1979) recognised that the exercise of representing meaning was a never-ending one which would only be accomplished in part, however he saw it as one worth pursuing and felt that even small successes would be valuable.

The term semantic modelling has been used to denote the overall activity of representing meaning. A multitude of different models have been proposed but despite their differences they have typically adopted a similar approach to the problem, described by Date (1986) as follows :-

(i) a set of semantic concepts are identified that can usefully represent information about the real world;

(ii) a set of symbolic objects are designed to represent the semantic concepts;

(iii) integrity rules are devised for the symbolic objects to ensure that the database is accurate and correct;

(iv) a set of operators are defined to manipulate the symbolic objects.

Date (1986) considered a data model to consist of the objects, rules and operators but thought that some developers had concentrated almost solely on the data structures to the detriment of the latter two aspects.

Of the many semantic models that have been developed, two which have been particularly influential are the entity-relationship model (Chen, 1976) and the RM/T model (Codd, 1979; Date, 1983).

The entity-relationship model was one of the first semantic data models to be proposed. Chen sought to include the advantages of the three basic models and designed the entity-relationship model to be a generalisation and extension of them. Central to the entity-relationship approach is the view that the real world can be modelled in terms of *entities* and *relationships*, these are the semantic concepts of the model. Chen defined an entity as being “a thing that can be distinctly identified”, for example a particular person or event. Entities can be classified into entity sets, which do not necessarily have to be mutually disjoint, and tests for set membership can be performed. A relationship was identified as being “an association among entities”. A relationship set was then defined as being a mathematical relation among a number of entities taken from one or more entity sets. As usual relationships can be one-to-one, one-to-many or many-to-many. Information about the entities and relationships is in the form of attribute-value pairs. An attribute is some quality or quantity which is observed or measured as a value taken from a value set, set membership for values would also need to be validated.

To assist in the database design stage Chen developed a diagrammatic notation, the entity-relationship diagram, a simple example from Chen’s paper is given in Figure 3.1. The diagram indicates that an employee works on a number of projects and a project has a number of workers involved in it.

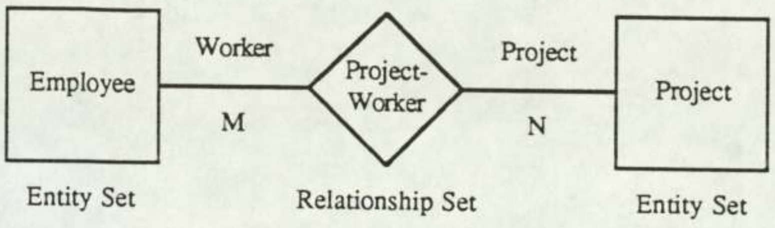

Figure 3.1 Example Entity-Relationship Diagram

A database would consist of information relevant to the entities and relationships of that part of the world being modelled. The entities and relationships would first be identified and represented in an entity-relationship diagram, the attributes and value sets could then be defined. The information about the concepts identified would then be represented using *entity* and *relationship relations*. Chen proposed that an entity relation would consist of information for a number of entities, each of the same entity type, measured over a number of attributes. The entities would be identified by a primary key consisting of either a single or a combination of attributes. A relationship relation would then associate one or more entity relations. The primary key of a relationship relation would be composed of the primary keys of the entity relations involved. Relationship relations could also have their own independent attributes.

The entity relation for the entity type EMPLOYEE from Figure 3.1 could be defined as in Figure 3.2.

|←Primary Key→|

Attribute	Employee-No	Name		Age
Value Set	Employee-No	First-Name	Last-Name	No-of-Years
Entity Tuples				

Figure 3.2 Entity Relation EMPLOYEE

If the entity relation PROJECT had a primary key PROJECT-NO then the relationship relation PROJECT-WORKER could be as shown in Figure 3.3.

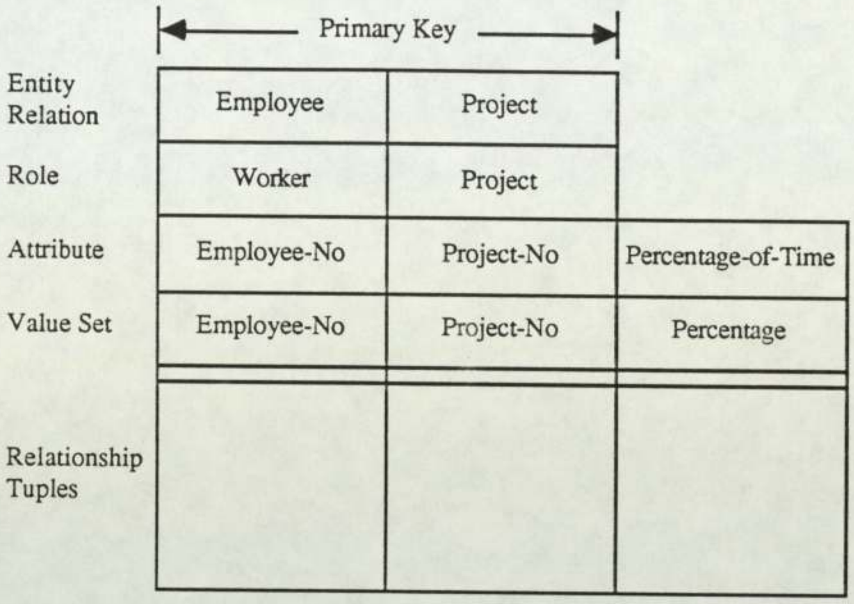

Figure 3.3 Relationship Relation PROJECT-WORKER

Chen's paper was largely concerned with the modelling aspects of the entity-relationship model and he only briefly discussed data integrity, information retrieval and data manipulation aspects. With regard to data integrity, testing for set membership has already been mentioned. Chen noted that some attributes could be drawn from a subrange of a value set, for example ages of employees as a subrange of all ages, a particular value could also be constrained by the value of another attribute, an employees tax value will be less than their salary value. Chen expressed an opinion that rules for retrieval, insertion, deletion and updating would be simpler and clearer when using the entity-relationship model but did not expand on this aspect to any great extent.

The RM/T model has been designed as an extension of the basic relational model. The original version was proposed by Codd (1979), since then a number of refinements have been made and the improved version has been described by Date (1983).

RM/T is also founded on the assumption that the real world can be modelled in terms of entities which can be classified into entity types. However in contrast to the entity-relationship model, a relationship is considered as being a special kind of entity. The constructs provided by RM/T allow a number of relationships to be represented.

In RM/T, entities and entity types are classified into one of three categories :-

(i) Characteristic

A characteristic entity performs a subordinate function to qualify or describe a superior entity upon which it is existence-dependent. Such entities were defined to represent the occurrence of repeating groups. For example, a purchase order will consist of quantities of a number of items. An entity type ORDER could be declared with a characteristic entity type ORDERLINE. For each entity of type ORDER there will then be a number of entities of type ORDERLINE, one for each item in the purchase order. RM/T allows characteristic entities to have further lower level characteristic entities to describe them.

(ii) Associative

Associative entities represent relationships between two or more entities that are in all other respects independent. Since associations are considered as being entities they can have characteristic entities to describe them and may also be part of other associations.

(iii) Kernel

Entities which exist independently and are neither characteristic nor associative are kernel entities.

All three categories of entity type can have information about them in the form of *properties*, cf. attributes in the relational model.

In RM/T entity types can form type hierarchies, that is an entity type can have a number of subtypes and may itself be a subtype of some supertype. Type hierarchies can be formed for all three classes of entity type but a hierarchy can only contain entity types of the same class, that is the subtype of a kernel entity type will also be a kernel entity type. Codd noted that the advantage of allowing such hierarchies was that properties of entity types could be declared at the most general level. Consider the entity type hierarchy in Figure 3.4.

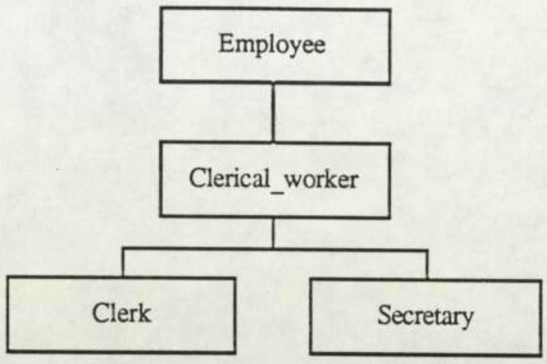

Figure 3.4 Example RM/T Entity Type Hierarchy

Those properties declared for employees will also apply to all subtypes, however properties specific to clerks can be declared lower down.

Information about entities is represented in *E-relations* and *P-relations*. For each entity type an E-relation is generated, this is a unary relation which will be used to record which entities of that type exist, the properties of those entities are recorded in P-relations. Codd proposed that entities would be identified by system controlled *surrogates*, the values of which would be hidden from the user. It is the surrogate values that are stored in the E-relations and which identify the properties in the P-relations. When information about an entity is entered a surrogate value is generated, it is not only inserted into the E-relation for that entity type but also into the E-relations of all supertypes. The properties are then entered into the appropriate P-relations.

Much of the description of the RM/T model was concerned with the modelling aspects of representing information about entites and the relationships between them. A number of integrity rules were described, these were developed to ensure that the database is maintained in a consistent state. A number of high level operators to manipulate the information to provide users with a variety of views of the database were also outlined.

From the published work of those developing semantic database models, the one aspect which has been predominant is that of developing constructs to represent the part of the real world being considered. The primary semantic concepts which the many models have sought to represent, albeit using different formalisms, have been objects, attributes and relationships among objects. Many of the papers have been at a purely theoretical level, with models being proposed and updated. It is only recently that database management systems based upon a semantic model have become commercially available. The initial use that semantic models were put to was as design tools. A schema to represent the real world would be designed using a semantic model and then transformed to one of the standard models, often the relational model. The entity-relationship model has emerged as the most popular although a number of extensions have been incorporated since Chen's initial proposals.

Those systems which have been implemented have concentrated on the data integrity aspects and operators to manipulate the information to answer user requests. As yet, intelligent database systems capable of making inferences from the information in the model have yet to get beyond an initial theoretical stage.

3.2.2 Semantic Networks

Other work on representing semantic knowledge has been carried out in the field of artificial intelligence, in particular with the development of semantic networks.

This representation was initially conceived by Quillian (1966), his intention was to use a network consisting of nodes and links to represent the semantics of English words. He wished to build a model of human memory based on the idea of associations, such that human-like use of the meaning of the words could be made. Each of Quillian's word concepts was made up of other words, with the organisation thus resembling that of a conventional dictionary. Since Quillian's work, research has continued in the use of semantic networks for representing the meaning of English text such that inferences can be made.

A semantic network, in its simplest form, consists of a collection of nodes interconnected by a set of arcs. Although originally designed to represent natural language concepts, it has been recognised that this type of representation is general enough to represent other forms of knowledge. Nodes can be used to represent objects or concepts of some kind, with arcs denoting binary relations between them. One use in particular has been for representing taxonomic hierarchies, that is hierarchically classifying classes of objects. Semantic networks have been used to represent various types of taxonomies, a number of which have been discussed by Brachman (1983). Most have been based upon the foundation of set theory and have used the concept of inheritance of properties. A hierarchical classification scheme has the opportunity to distribute properties throughout the levels of the taxonomy offering an efficient storage scheme. The classification of entity types in semantic database models can be regarded as semantic networks.

3.3 Statistical Data Models

The notion of representing information about statistical data in the form of a data model is consistent with the development of statistical database management systems. Such systems have been necessitated by the need to handle increasingly large and complex data sets. The data management facilities of most general statistical packages were seen to be deficient in such circumstances and a database approach appeared to offer a solution. Systems based on the relational model have been predominant, for example the RAPID system which has been developed at Statistics Canada (Hammond, 1983).

In order to effectively administer the data, both the users and the systems software require a reasonably detailed description of the contents of the database. This information has been in the form of *metadata*, that is data about data. McCarthy (1982) described metadata as being "systematic descriptive information about data content and organisation". There is as yet no consensus of agreement as to what this metadata should constitute, although there is of course endless scope.

A paper by Lundy (1984) proposed that the definition of metadata could be approached from two directions: a functional perspective based on proposed use; an operational outlook founded on the type of information incorporated in database manipulation systems. The main purposes for representing metadata have been identified as being to enhance documentation, retrieval and display features. In storing data with a complex structure, for example hierarchical, metadata is needed by the system to locate data which has been requested by a user. Users will also wish to document the source and content of a database when storing large and numerous data sets. The presentation of tables and graphics can be greatly enhanced with the accompaniment of additional non-essential information. McCarthy noted that metadata, in contrast to data, would be primarily textual.

In considering the role of metadata, it has chiefly been seen as an aid to data management. This is in accordance with its role in general database systems, although different types of metadata specific to a statistical application have been identified. Metadata has been used to verify the correctness of some numeric operations but its use for statistical purposes is something that has not been considered to any great extent.

3.4 Research Objectives

The objective of the research is to implement a prototype system as a means of exploring the feasibility of semantic modelling as an approach to knowledge based statistical software. This will entail developing a model to represent some of the semantic meaning of statistical data and devising a set of semantic checks which can be applied to validate some of the assumptions underlying the use of statistical methods. The checks will be intended to see if a test is suitable for the data that has been specified, that is will the application of a test be sensible. It is not intended to confirm that a test is appropriate for the hypothesis of the user.

It is possible to neither represent all of the semantic meaning about the data nor to verify all of the requirements regarding the proper use of a test. For practical purposes the amount of semantic knowledge that can be represented is limited, hence the checks that can be performed are restricted by the knowledge available. The problem is that of deciding what knowledge to represent. A system which requires a vast amount of information about the data to be declared will not be popular with users, conversely the level of checking must be sufficient to make the use of such a system beneficial. Also the marginal return on increasing the amount of knowledge in the model diminishes as more knowledge is added, once an optimal point is reached the increase in the level of checking that can be done is low in proportion to the amount of extra knowledge required.

It is hoped that if the approach appears to be workable, some conclusions regarding the content of a semantic model can be drawn as a result of the work carried out.

Chapter 4
Introduction to the Prototype System

A prototype system has been developed using VAX Pascal to explore the feasibility and practicability of a semantic modelling approach to knowledge based statistical software. The system has been designed as an enhanced variant of the command driven general purpose packages that are currently put to widespread use. The enhancements that have been incorporated aim to reduce the amount of misuse of the statistical facilities provided. By representing more semantic knowledge about the data the system is better able to validate a request to perform a statistical operation.

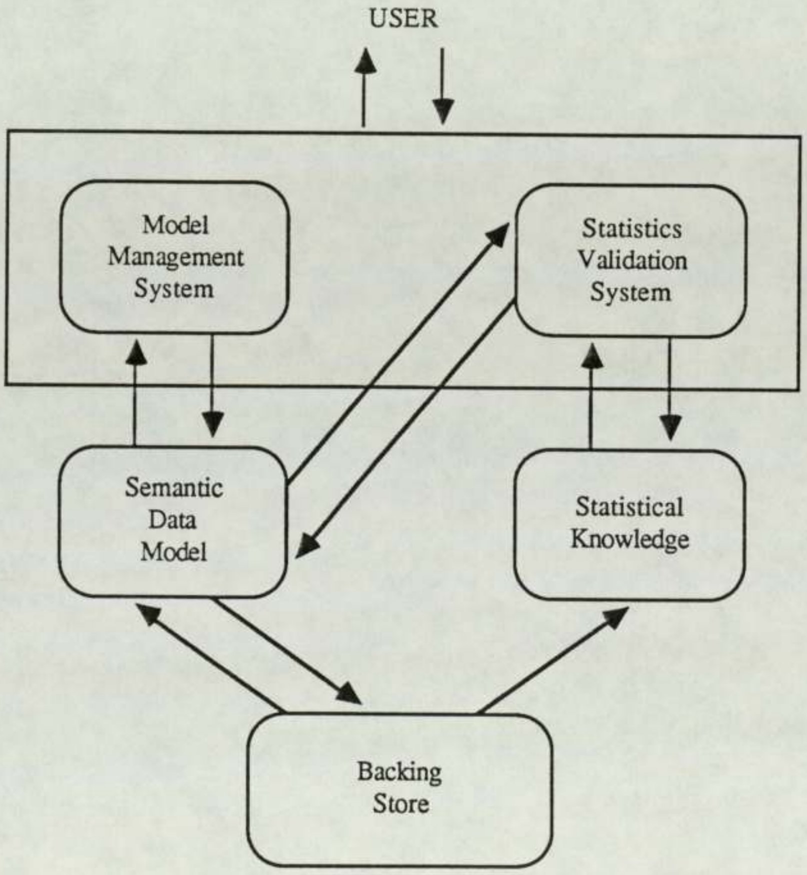

Figure 4.1 Main Components of Implemented Prototype

In developing the system a number of simplifications and limitations have been made but these do not affect the underlying approach. Figure 4.1 illustrates the main components of the implementation and their interaction. The system itself consists of two main sections, a *model management system* and a *statistics validation system*, the information that it operates on is the *semantic data model* and the *statistical knowledge.*

The user declares and queries the knowledge represented in the semantic data model using commands that are processed by the routines comprising the model management system. The semantic data model is represented by the system as a number of Pascal data structures, these structures are extended as more semantic knowledge and data is added to the model and searched whenever information is required. Long term storage is achieved by maintaining a copy of the data model in the *backing store*. Whenever it is necessary to add information to the data model, the model management system also updates the copy in the backing store. When the system is invoked, the data structures representing the semantic data model are initialised from any information in the backing store, which for the purposes of the prototype are Pascal text files.

The statistical knowledge consists of the requirements that must be met for the statistical methods to be applicable. When the user issues a command to perform a statistical operation on a number of arguments, the statistics validation system searches the semantic data model to examine the knowledge that has been declared about the arguments to determine if the requirements can be satisfied. The statistical knowledge is represented in a number of Pascal data structures, which are initialised when the program is invoked from information in the backing store. The method devised to represent the statistical knowledge results in a program that is not rule-based but not wholly procedural either.

The command language that has been developed for the prototype has commands which follow the general syntax given below.

< command name >   < list of arguments >

The command name is parsed by the main program, if it is found to be valid a call is made to the relevant procedure in the model management system or the statistics validation system. These procedures read in the arguments of the command and either report any error that has been made or perform the required actions.

There are also a number of procedures performing auxiliary functions. The procedure *gettoken* is called whenever an item of input is required by the main program or a procedure processing a command. Gettoken reads in the next lexical token and determines if the input is a reserved keyword, an identifier, a numeric value, a special symbol or erroneous. Appropriate values are assigned to the record variable *token* to reflect the input found. Any errors that are found are reported with a call to the *reporterror* procedure. A parameter is passed identifying the type of error found such that the routine can produce a suitable error message, a character string is also passed which may be used to further pinpoint the source of the error.

Those aspects of the system which can be thought of as forming the user interface have only been developed to a limited extent, but they are sufficient for the purposes of developing and testing the prototype. With regard to the model management system, the error checking is thorough and the messages produced are adequate but there is no help facility to give the required syntax of a command or to explain the meaning of the terminology used. The statistics validation system does explain the results of validating the use of a statistical operation and can also provide information about the requirements that must be met for the operation to be appropriate. By default both sets of information are provided, the user can however issue the command NOEXPLAIN to indicate that only the results are required, the command EXPLAIN can be issued to return to the default setting.

It is conceived that by adopting a semantic modelling approach to knowledge based statistical software it would be possible to produce a general purpose statistical system able to offer support over a wide range of statistical tests and techniques. For the prototype version it was necessary to limit the areas of statistics that could be covered, those areas chosen were measures of association and tests of location. The choice was made to develop the prototype for these areas because :-

(i) they are commonly used, introductory statistics courses are included in a great many degree programs and there is a tendancy for people to use what is familiar to them;

(ii) superficially they appear simple in the literature and are easy to calculate using a statistical package. Users may have a false impression that they understand about the statistic and proceed to use it independently. This is less likely to occur with those advanced techniques where it is more readily apparent that assistance is required.

As a result, measures of association and tests of location are commonly and unwittingly misused and therefore seemed to provide a suitable testing-ground for the approach. The system does not actually compute the result of any measure of association or test of location where usage has been validated, this task does not however present any problems.

An example of the system running is given in Appendix A. The execution trace illustrates the use of a number of the model management commands to display the contents of a data model that has been declared, this is followed by an example of the operation of the statistics validation system.

Chapter 5

The Semantic Data Model

5.1 Semantic Concepts

The first task was to identify the concepts which would be used as the basis of the semantic knowledge to be represented in the model. The semantic models proposed for general purpose database management applications have been founded upon the concepts of entities, attributes and relationships. These concepts have been proposed as a means of representing the real world in terms of objects and their properties. A similar approach and notation has been adopted but the model has been developed to incorporate information which can represent the statistical nature of data. These concepts have been built onto the framework of a relational schema. Designing a system based on a basic database model is in accord with the need to develop statistical software with data management facilities, akin to those of database systems. Haux & Jöckel (1986) argued that there was a need for intelligent statistical systems which could combine, in an integrated manner, both data management and data analysis functions. The reason for selecting the relational schema as the foundation for the model is that it has emerged as the one most commonly used for general database management systems and it has also been chosen by the majority of those developing statistical databases.

In deciding what semantic knowledge to represent about the data, a starting point is to consider :-

(i) what class or type of object the data is being measured for;

(ii) which specific instances or objects the data is pertaining to;

(iii) what property of the object is the data depicting, that is some quality or quantity is being represented or measured, e.g. a height or an examination result;

(iv) how the quality or quantity is being represented, for example an examination result could be represented as a percentage or as a grade.

The fundamental data structure of the model has been termed a *dataset*. This is a rectangular construct which consists of data measured for a number of entities, instances of an entity type, over a number of attributes, cf. a table in the relational model. In order to include the required semantic information it was necessary to represent knowledge about datasets, entity types, entities and attributes.

Entities can be identified as being of a specific entity type and a hierarchical taxonomy has been used to classify these entity types, relationships between entity types can then be examined. Only two types of links have been implemented, *generic* and *nongeneric*, to allow one entity type to be declared as being a specialisation of another. The reason for including an entity type taxonomy in the data model is that it provides an efficient method of representing knowledge that different entity types are in fact similar. Some statistical methods will only produce meaningful results if the data involved is measured for objects that are alike. In the semantic modelling literature little attention has been paid to the use of entity taxonomies and the variety of relationships that should be possible. Semantic network applications have also been satisfied with denoting that one type is a generalisation of another. In both these areas

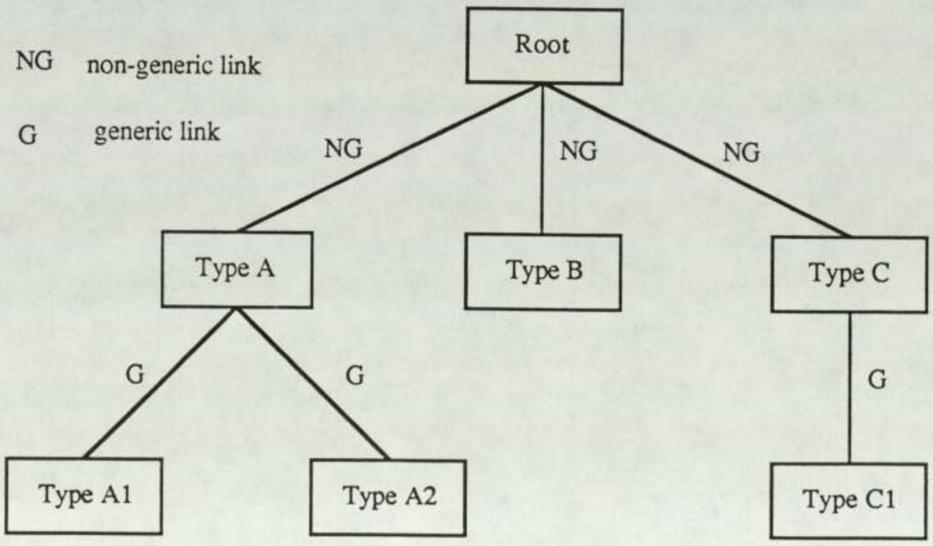

Figure 5.1 Use of the Entity Type Taxonomy

the purpose of the taxonomy has primarily been to allow for the inheritance of properties rather than to represent what the objects are. The use of the two links that have been implemented allows a number of sub-trees of 'like' entity types to be constructed below the root node. Figure 5.1 illustrates that there are three distinct types of entities, with type A having two subtypes and type C one.

The data in a dataset will be for a number of entities which are instances of an entity type. This type will be represented in the entity taxonomy. For each dataset, it would be desirable to be able to identify these instances. This has been achieved by following the usual relational database convention of declaring key fields. One or a combination of the attributes can be declared as constituting a unique key to identify the instances in a dataset. The use of keys makes it possible to compare the instances from a number of datasets to see if they coincide. This is an important property of the data to be able to recognise, for some statistical methods only paired or related samples can be used whereas with other methods independent samples should be treated in a different manner to those which are paired or related. It is optional for a key to be declared for a dataset and the concept of secondary keys has not been implemented.

Attributes are used to record the properties of entities, in a basic relational model they are drawn from *domains*. A domain is usually defined as being a set of atomic values from which the data is drawn. This is concerned with the type and range of possible values and has analogies with the programming language concept of a data type. Attributes which record the same property but using a different notation would be specified as being drawn from different domains, for example heights in centimetres and heights in inches. This loss of information is of little consequence in a general purpose database system as domains are primarily conceptual, the main use of which is for data validation purposes. For a statistical application, information about the domain of an attribute can be more usefully represented as several items of information. In the model that has been developed these items are concerned with 'what' the property is and 'how' it is being denoted.

It is important to know what property is being represented by data, comparing the data of different attributes would be meaningless unless it was homogeneous. By separating what the property is and how it is being denoted, it is possible for the system to differentiate between the following three cases :-

(i) the property being represented is the same and it is being represented in the same way, thus indicating homogeneous data;

(ii) the property being represented is the same but it is being represented in different ways, it is possible to obtain homogeneous data if it can be converted into a common method of representation;

(iii) the property being represented is different, the data cannot therefore be made homogeneous.

The 'what' part, termed the *attribute type*, has been represented as a character string, e.g. "height". In considering 'how' the data is being represented, this is concerned with the measurement aspects, e.g. measured using inches. Representing this semantic knowledge has been achieved by including in the model, information about the level of measurement and the measurement scheme used for each attribute. A committee of the British Association for the Advancement of Science debated the subject of measurement and identified a classification of scales of measurement, as described by Stevens (1946). The classes are determined by both the manner in which the data is measured and by the formal mathematical properties of the scales. The importance of these scales is that the mathematical and statistical operations that can be meaningfully applied to data are dependent upon their scale of measurement.

The scales - nominal, ordinal, interval and ratio - were identified as follows :-

(i) a nominal scale uses symbols to denote group membership, this is the lowest level of measurement;

(ii) an ordinal scale has the additional property that the groups can be ordered such that a greater-than relationship can be identified between them;

(iii) with an interval scale the difference between any two values can be determined;

(iv) a ratio scale has the property of having a true zero such that the ratio of any two values is independent of the unit of measurement.

These scales have been adopted for the data model. In addition it has proved useful to distinguish as a separate category any data which is represented using rank values, ranks are usually classified as being a type of ordinal scale. For attributes that have nominal, ordinal, interval or ratio scale data there will in addition be information about the unit of measurement used to record the data, this is not required for rank level data. The distributional properties of data is also very useful information to have. It is possible for the user to declare if they know that interval or ratio level data is drawn from a normally distributed population. No other distributional aspects have been implemented.

The knowledge about how the data has been recorded is completed with a description of the measurement scheme that has been used. There are descriptions of all the schemes that have been declared to the system in a measurement directory. The measurement schemes can be classified as being either qualitative or quantitative. A qualitative scheme will consist of a closed or an open set of categories, that is a finite or an infinite set of possible values. Closed sets are either unordered or ordered and a list of possible values is stored. For each quantitative measurement scheme the upper and lower bound of possible values will be stored. By having knowledge of the set or range of possible values, data validation functions can be performed as the data is entered. For all attributes with interval and ratio level data there will be an associated quantitative measurement scheme, in the case of rank data there will be no such scheme. Ordinal level data could also have been measured using a quantitative scheme, this is to allow for data such as IQ scores, or alternatively the values may have been drawn from an ordered set of qualitative categories. All nominal data is qualitative and could be represented as values drawn from open or closed sets.

The level of measurement and measurement scheme combinations that can be declared for an attribute are summarised in Table 5.1.

Level of Measurement	Measurement Scheme Restrictions
Ratio	Quantitative
Interval	Quantitative
Rank	
Ordinal	Quantitative
Ordinal	Qualitative, Closed Set, Ordered
Nominal	Qualitative, Closed Set
Nominal	Qualitative, Open Set

Table 5.1 Measurement Level and Measurement Scheme Combinations

The last aspect of semantic knowledge that has been represented is a directory containing information on how to convert data from one measurement scheme to another. It is possible for data to be converted from one quantitative measurement scheme to another, to convert quantitative data into ordered qualitative categories or to convert one set of closed categories to another. Of the quantitative to quantitative conversions that are possible, only linear transformations of the form $y = ax+c$ have been implemented, where c will be zero for all ratio scale conversions. To categorise a quantitative set of data a range of values are mapped onto one of the categories of the qualitative measurement scheme. When converting data from one category set to another, each category in the source scheme is mapped onto a category in the target measurement scheme.

Users will declare the semantic knowledge about their sets of data and a semantic data model consisting of an entity taxonomy, a dataset directory, a measurement directory and a conversion directory will be built. The data structures to store this information and the routines to build the model are described in the remaining sections of this chapter.

5.2 Symbolic Objects

This section describes the Pascal data structures that are used for the internal representation of the semantic data model.

5.2.1 Entity Type Taxonomy

The entity taxonomy is represented as a dynamic data structure using a node for each entity type that is declared. An entity type will have one immediate supertype, of which it is a specialisation of, and could subsequently have any number of immediate subtypes. The representation was required to support operations to traverse up and down the taxonomy in addition to the need to add a new entity type as a subtype of an existing type.

A diagrammatic representation of the structure used to denote an entity type is given in Figure 5.2.

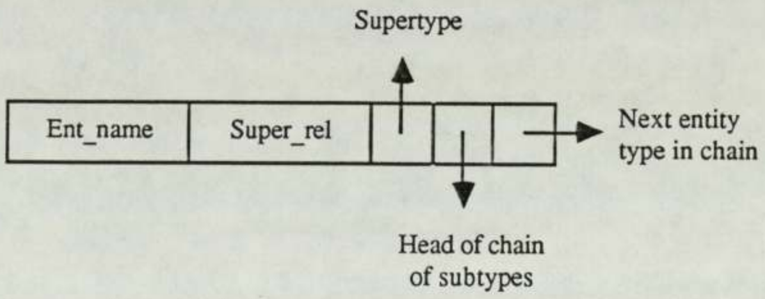

Figure 5.2 Entity Type Node

Each entity type node will contain the following information :-

(i) ent_name - a character string that identifies the entity type;

(ii) super_rel - an enumerated type to indicate the relationship between the entity type and its immediate supertype, it will take the value *nongeneric* or *generic*;

(iii) superpointer - a pointer to the immediate supertype node;

(iv) subpointer - a pointer to the head of a chain of entity types which have this entity type as their immediate supertype, this pointer will have the value NIL if there are no subtypes;

(v) nextpointer - a pointer to the next entity type in the chain of entity types which share a common supertype.

An example of how an entity taxonomy would be represented is given in Figure 5.3. The root node, which is predeclared by the system, has been declared as the supertype of three entity types (EntA, EntD and EntE), the nodes for these entity types have been chained together. Two subtypes have been declared for EntA and one for EntE.

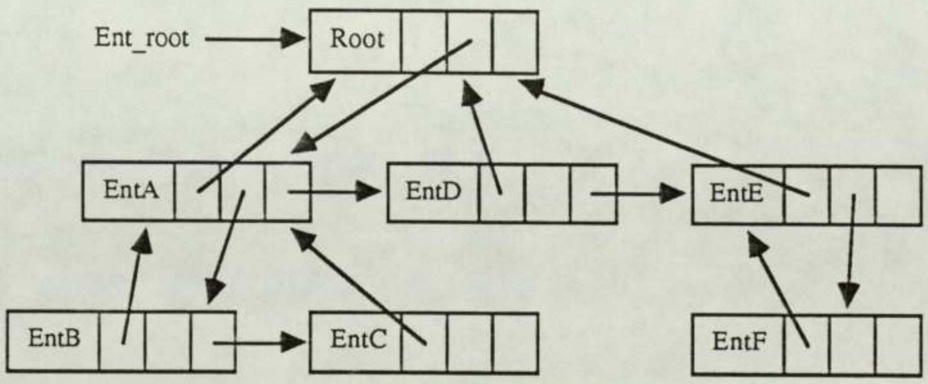

Figure 5.3 Representation of an Entity Taxonomy

5.2.2 Measurement Directory

A binary tree structure has been used to represent the measurement directory which is indexed by the measurement name. It was decided to use this type of data structure because it provides an efficient method of searching for a particular measurement scheme entry. It is beneficial to have a single directory containing information about both qualitative and quantitative measurement schemes. The use of a Pascal variant record solves the problem of needing to store different information about each type of measurement scheme.

The common part of the measurement node will represent the following information :-

(i) measname - a character string that identifies the measurement scheme;

(ii) leftp, rightp - pointers to build the binary tree;

(iii) meas_type - an enumerated type to indicate the type of measurement scheme being represented, this acts as the *tag field* for the node and will take the value *qualmeas* or *quantmeas.*

If the meas_type field has the value qualmeas then the measurement node will be as shown in Figure 5.4.

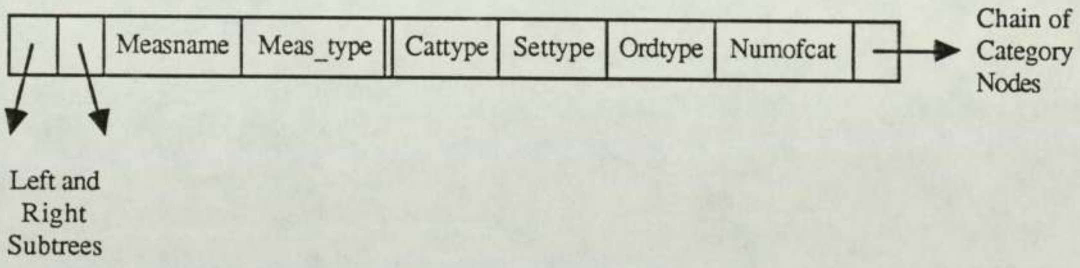

Figure 5.4 Qualitative Measurement Node

The additional information that is stored for a qualitative scheme is as follows :-

(i) cattype - an enumerated type which indicates the type of the data, the user may have represented the category labels as numeric values or as character strings, this field will take the value *identifier* or *numeral*;

(ii) settype - an enumerated type which will either have the value *openset* or *closedset* to designate a set of possible values which is infinite or finite;

(iii) ordtype - if there are a finite set of possible values, the members may be *unordered* or *ordered*;

(iv) numofcat - in the event of a closed set this field indicates the number of categories;

(v) cathead - a pointer to the head of a chain of category nodes.

For closed sets, each permissible value is stored in a category node. For category sets where there is an underlying order present, the values are stored in ascending order from the head to the tail of the chain.

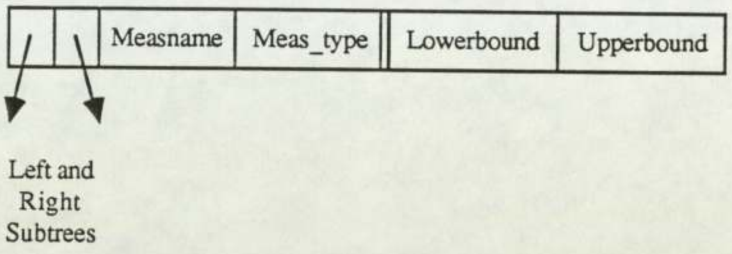

Figure 5.5 Quantitative Measurement Node

If the measurement scheme being represented is quantitative, as denoted by a meas_type value of quantmeas, the variant record will be in a form as illustrated in Figure 5.5. The information specific to a quantitative measurement scheme is represented as follows :-

(i) lowerbound - a real value indicating the minimum of the allowable range;

(ii) upperbound - a real value indicating the maximum of the allowable range.

5.2.3 Dataset Directory

The dataset directory is organised as a binary tree which can be searched for alphabetically by dataset name. This directory contains all of the information relating to the datasets and their associated attributes. For each dataset that has been declared there will be a node in the tree containing information pertaining to the dataset as a whole, such a node is shown in Figure 5.6.

The information in the dataset node is as follows :-

(i) ds_name - a character string to identify the dataset;

(ii) leftp, rightp - pointers to build the binary tree;

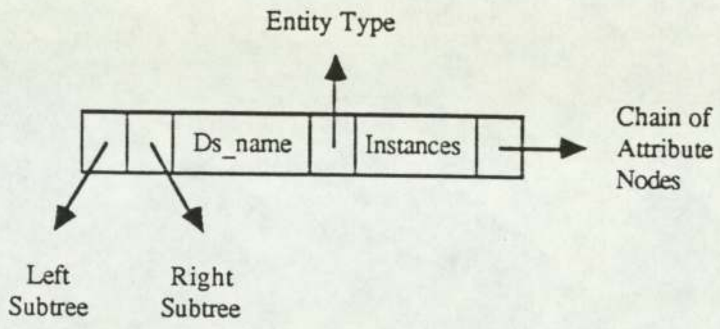

Figure 5.6 Dataset Node

(iii) ent_type - a pointer to an entry in the entity type taxonomy, the entities in the dataset will be instances of this entity type;

(iv) instances - the number of instances that have been entered for the dataset;

(v) attchain - a pointer to the head of a chain of attribute nodes.

A node will be generated for each attribute that is declared for the dataset. The attribute nodes will be stored in the order in which they are declared. Key attributes are declared first in their order of significance in the key. A dataset as a whole is referred to as *<dsname>* whereas a specific attribute is referenced as *<dsname>.<attname>*, whereupon the chain is searched for the required node. The attribute nodes are as in Figure 5.7.

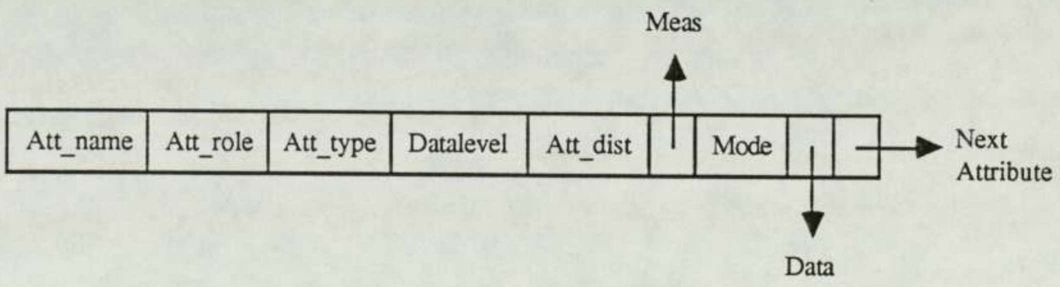

Figure 5.7 Attribute Node

The knowledge represented in the attribute node is as follows :-

(i) att_name - a character string to identify the attribute;

(ii) att_role - an enumerated type value, if the field is set to *key* it signifies that the attribute forms part of the key of the dataset, otherwise the value will be *non_key*;

(iii) att_type - a character string to designate what property the attribute is representing, e.g. height;

(iv) datalevel - denotes the level of measurement of the data which will be *nominal*, *ordinal*, *rank*, *interval* or *ratio*;

(v) att_dist - an enumerated type whose value is set to *normaldist* if the user has declared a knowledge that the data is drawn from a normally distributed population;

(vi) meas_p - a pointer to an entry in the measurement directory to indicate which measurement scheme has been used to record the data, this field will be NIL for any rank level data;

(vii) mode - identifies the type of the data which is stored for the attribute, *identifier* signifies alphanumeric data otherwise *numeral* indicates numeric data;

(viii) char_p, num_p - the data for the attribute is stored in an array which is referenced by a pointer in the attribute node, the mode value acts as the tag field so that the pointer is of the appropriate type, i.e. to a character or numeric array;

(ix) next_att - a pointer to the next attribute node in the chain.

The user's original data, as they enter it into the system, will be either alphanumeric or numeric. All quantitative and rank data is stored as it is entered in a numeric array. For qualitative data, the user may have represented the category labels using either alphanumeric or numeric values. For qualitative data from closed sets, the index of the category value in the value set is stored instead. When a category value is entered it is necessary to search through the list of permissible values to validate set membership. Although the index values are not stored, the category value at the head of the chain is regarded as having the index value 1. Since categories for ordered sets are stored in ascending order from head to tail, the index value preserves the underlying order. The advantage of storing the index value is that all data likely to be used for statistical operations is stored in numeric arrays, with only qualitative data from open sets having alphanumeric labels being stored in character arrays. Being able to handle the data in a consistent manner is also an advantage with regard to performing data conversions. All of the data to be converted will be numeric and the information relating to conversions involving qualitative measurement schemes can be stored with respect to the index numbers of the categories.

To simplify the implementation of the prototype, the data has been stored in arrays of size 50. For a practical application a storage system able to cope efficiently with large and small sets of data would be required.

5.2.4 Conversion Directory

Each entry in the directory contains the information on how to convert data from one measurement scheme to another. A conversion scheme is identified by its name, *from_to*, which is formed by concatenating the names of the source and target measurement schemes. The directory is organised as a binary tree which is searched for by name, a conversion node is shown in Figure 5.8.

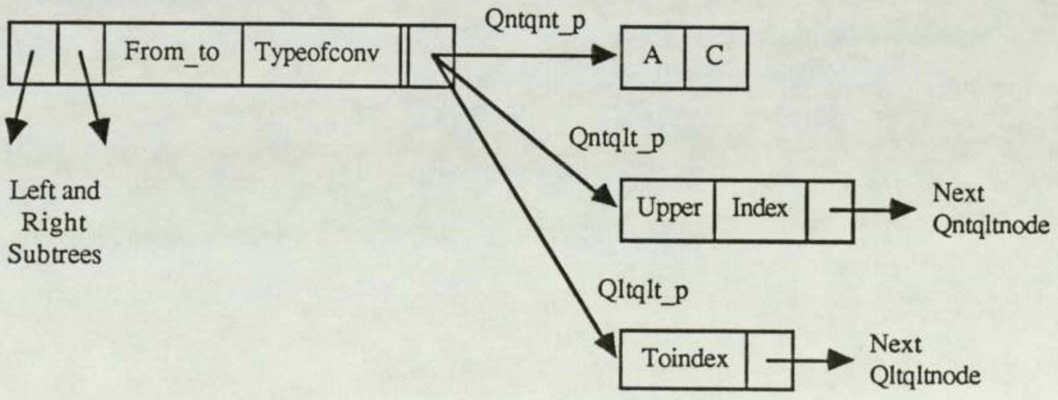

Figure 5.8 Conversion Node

The tag field *typeofconv* identifies the type of conversion as being *qnt_qnt*, *qnt_qlt* or *qlt_qlt*. The type of the pointer to the information on how to perform the conversion is determined by this value, it has the respective types *qntqnt_p*, *qntqlt_p* or *qltqlt_p*.

To represent how to convert from one quantitative measurement scheme to another it is necessary only to store the constants from the equation

$$\text{target} = a * \text{source} + c.$$

A quantitative to qualitative conversion involves splitting the range of the quantitative measurement scheme into a number of sub-ranges. These sub-ranges are mapped onto the categories of the qualitative scheme in a form as shown below.

$$\begin{array}{ccccc} \text{lowerbound} & \leq & \text{toindex}_i & \leq & \text{upper}_1 \\ \text{upper}_1 & < & \text{toindex}_j & \leq & \text{upper}_2 \\ : & & : & & : \\ \text{upper}_n & < & \text{toindex}_p & \leq & \text{upperbound} \end{array}$$

A chain of nodes are created which contain the *upper* and *toindex* values for each sub-range, these nodes are stored in ascending order of quantitative sub-ranges.

Converting one qualitative measurement scheme to another involves mapping each source category label to one in the target category scheme. This can be viewed as below.

$$\begin{array}{ccc} \text{fromindex}_1 & \equiv & \text{toindex}_i \\ \text{fromindex}_2 & \equiv & \text{toindex}_j \\ : & & : \\ \text{fromindex}_n & \equiv & \text{toindex}_p \end{array}$$

Each node in the chain contains the toindex values which relate to the fromindex categories in the order in which they are stored in the measurement directory entry.

5.3 Semantic Data Model Manipulation Commands and Procedures

A minimal set of commands have been implemented to define, add data to and query a data model. The system checks the syntax and semantics of user issued commands and outputs error messages as appropriate. For a commercial system the interface would need to be substantially further developed. In particular, to assist users in defining a knowledge base by explaining the meaning of terminology. For example, as it stands it is necessary to declare the level of measurement of an attribute, for a novice user who was not able to declare the level it would be desirable to be able to question the user such that the level could be deduced.

5.3.1 Entity Types

The command ADDENT is used to add an entity type to the taxonomy.

```
ADDENT <newenttype> <superenttype> <relationship>
```

The system searches the taxonomy to ensure that the new entity type, newenttype, has not already been declared and that the specified supertype, superenttype, has. The only other check that is carried out is to see that a valid relationship has been given. If no error has occured, an entity type node is generated and is then added to the taxonomy at the head of the chain of subtypes for the superenttype node.

The entity types that have been declared in the taxonomy can be displayed using the SHOWENTDIR command.

```
SHOWENTDIR <enttype>
```

This displays the part of the taxonomy from the enttype node downwards, to display the whole taxonomy the entity type ROOT can be specified.

5.3.2 Measurement Schemes

The user can declare a new measurement scheme to the system by issuing the ADDMEAS command.

```
ADDMEAS <measname> <meastype>
```

The system will initially check that the measname scheme does not already exist in the directory and that a valid meastype value of *qualmeas* or *quantmeas* has been given.

If no error has been found the appropriate procedure, *getqualinfo* or *getquantinfo*, is called to complete the process of declaring the measurement scheme.

Alternatively, if during the course of some other operation, for example declaring an attribute, the user enters the name of a measurement scheme which is unknown to the system, the user will have the option of adding it to the directory. Should the user wish to do so, the appropriate procedure will be called to complete the operation.

The procedure getqualinfo questions the user to complete the information about the qualitative measurement scheme being declared. One of the parameters, *measspec*, conveys to the procedure a knowledge of what is already known of the qualitative scheme, thus avoiding asking the user unnecessary questions. The possible values that will be passed are :-

(i) ordqual - an ordered set of closed categories;

(ii) unordqual - an unordered set of closed categories;

(iii) orddich - an ordered set with two categories;

(iv) unorddich - an unordered set with two categories;

(v) dich - a closed set with two categories which may be unordered or ordered;

(vi) qual - nothing is known of the measurement scheme other than it is qualitative.

When the user has issued the `ADDMEAS` command the parameter measspec will be passed the value qual.

If the measspec parameter has the value qual then the procedure getqualinfo will ask the user if the set of categories is open or closed, since for all other measspec values it is known that there is a closed set. For closed sets of categories where the ordering is unknown, the system will query the user for the required value. For all types of qualitative scheme the user will be questioned as to whether the categories are represented with alphanumeric or numeric labels.

For measurement schemes with a closed set of categories the procedure *getqualclasses* is called. The user is prompted to enter the category labels one at a time. For each new label, the list of category nodes of previously declared labels is searched to ensure uniqueness within the scheme before a new node is generated and added to the tail of the list. The procedure ensures that at least two possible labels are declared and in the case of a dichotomy the process is halted once two valid labels have been entered, the measspec parameter is passed for this purpose.

The procedure getquantinfo obtains the lower and upper bounds of the permissible range of values for the quantitative measurement scheme. The upperbound should be greater than the lowerbound and the keywords *min* and *max* can be used to specify the extents of the range of real values, these have been given artificial values.

Once the measurement scheme has been correctly declared it is inserted into the appropriate place in the tree.

Two commands have been implemented to view the knowledge stored in the measurement directory. The `SHOWMEASDIR` command displays the names of each scheme declared in the directory, in alphabetical order, along with their meas_type values. For more detailed information about a specific measurement scheme, measname, the `SHOWMEAS` command can be used.

```
SHOWMEAS <measname>
```

For a qualitative scheme the set and datatype values are displayed, in the case of a closed set the list of value labels is also given. The lower and upper range values are given in the event of the measname scheme being quantitative.

5.3.3 Datasets

A new dataset is declared by issuing the `ADDDS` command.

```
ADDDS <dsname> <enttype>
```

The dataset directory is searched to ensure that the new dsname is unique. The command indicates that the new dataset will record data about instances of the entity type enttype, the taxonomy is searched to check that the specified entity type has been declared. If no errors have occured, a dataset node is generated and added to the dataset directory in the appropriate place.

The dataset directory can be queried with the `SHOWDSDIR` command. This displays the name of each dataset, in alphabetical order, with their corresponding ent_type and instances values.

5.3.4 Attributes

Once a dataset has been declared, the attributes for it can be entered using the `ADDATT` command.

```
ADDATT <dsname>
```

Having verified that the dataset dsname is present in the directory, for which no attributes have currently been declared, the system prompts the user to enter the information about each attribute one at a time, verifying the entry for one before prompting for the next. The information for any key attributes is initially requested and then that for those which do not form part of the key. The information for each attribute is in a form as shown below.

```
<attname>   TYPE = <typename> LEVEL = <datalevel>
            MEAS = <measname> NORMAL
```

The attname value is given first and the remaining items can be given in any order as required. The list of currently declared attributes for the dataset is checked to ensure that the new attribute name is unique. For each attribute that is declared, the TYPE and LEVEL values should be given, the other items may or may not be required.

For nominal, ordinal, interval and ratio level data a measurement scheme for the attribute must be given. The system will search the measurement directory to see if the specified scheme has been declared. If it has, the function *meastypeOK* is called to see if it conforms to the level of measurement value given for the attribute. Alternatively, the user is asked if they wish to add the measurement scheme to the directory. If the user does not wish to do so then they are asked to re-enter the information for the attribute, it was decided to take this course of action for simplicities sake. Should the user wish to declare the new measurement scheme the relevant procedure to do so is called. For nominal level data the getqualinfo procedure is called with a measspec parameter of qual. Ordinal level data however could be measured by a qualitative or quantitative measurement scheme. The user is queried as to which and in the case of the former the getqualinfo procedure is called with a measspec value of ordqual. For all quantitative data, be it ordinal, interval or ratio, the getquantinfo procedure is called. The NORMAL field is optional for interval and ratio level data.

Once it is verified that the information for the attribute is complete and correct, a new attribute node is generated and added to the tail of the chain of attribute nodes. At this point in time, the array to contain any future data is not generated, i.e. the char_p or num_p pointer field is set to NIL.

For a particular dataset, the attributes that have been declared for it can be displayed with the `SHOWATT` command.

```
SHOWATT <dsname>
```

The attributes are displayed in the order in which they were declared and the information associated with each is shown.

5.3.5 Instances

Once the attributes have been declared for a dataset the actual data items can be entered with the `ADDINST` command.

```
ADDINST <dsname>
```

If no instances have been declared for the dataset, the arrays to store the data are generated, otherwise the new data is added to what has previously been entered. The user is prompted to enter the data an instance at a time, one value for each attribute. The data is checked to see that it conforms to what is expected, i.e. that the data type is correct and the value is in accord with any measurement scheme specifications. In order to check a value which represents a label from a qualitative measurement scheme with a closed set of categories the *catsearch* procedure is called. For a valid label the index number is returned, so that it can be stored in the data array, otherwise a 0 is returned signifying an eroneous label. The number of items for each attribute is *noelements* and the new items are stored in array locations noelements+1.

If the instances in the dataset are identified by a key, then once a data item has been validated and stored for each attribute, it is necessary to check the uniqueness of the new key. For datasets whose instances are identified by a key it has been found useful to store the instances in key order. Thus when adding a new instance to an already sorted list, the key can be checked for uniqueness at the same time as inserting it into its correct location. The procedure *sortinstances* works through the data to find the position, *newpos*, where the new instance, currently stored in position noelements+1, should be inserted. If the key is found to be a repeated value then an error message is output, the data ignored and the value of noelements is not incremented. Otherwise the system moves down by 1 the data items from newpos to noelements and inserts the new data for each attribute. For simplicity it is assumed that the number of instances does not exceed the number that can be stored in a data array.

Having dealt with the most recently entered instance, the user is prompted to enter the next. By dealing with one instance at a time a better description of any errors made can be given.

The instances which have been entered for a dataset can be displayed using the `SHOWINST` command.

```
SHOWINST <dsname>
```

For data items which are stored as the index to a category label from a qualitative measurement scheme, the actual label is retrieved from the entry in the measurement directory and is displayed instead of the index.

5.3.6 Conversion Schemes

The user does not volunteer to the system how to convert from one measurement scheme to another, as with the other components of a data model, but the system calls the procedure *getconvinfo* as and when the conversion directory does not contain an entry which is required. Before proceeding, the user is asked if they know how to do the required conversion. If it is possible, the system examines the measurement schemes participating, *frommeas* and *tomeas*, to identify the type of conversion involved and calls the appropriate procedure to obtain the required information.

For a qnt_qnt conversion, the user is asked for a non-zero multiplying factor and a constant term for the equation $tomeas = a*frommeas+c$.

In the case of a qnt_qlt conversion the user is asked for the boundaries of the quantitative sub-ranges, in ascending order, and is required to enter the corresponding qualitative category label for each. The system guarantees that the entire range of the quantitative measurement scheme is covered and that valid category labels are given. For the last sub-range, the keyword *upper* can be used to signify the upper bound of the quantitative scheme instead of the numeric value.

Information on how to perform a qlt_qlt conversion is obtained by prompting the user with the frommeas category labels in turn and requesting the corresponding tomeas category labels for each.

For qnt_qlt and qlt_qlt conversions, the qualitative labels entered by the user for the tomeas scheme are validated by calling the *catsearch* procedure.

5.4 File Storage of the Semantic Data Model

The information entered into a data model is stored in a number of Pascal text files. As the information is declared and added to the components of the data model the files are generated by the system and updated. This is done such that they always reflect the current content of the data model. When the system is invoked at the start of a run, the information is loaded from the files into appropriate Pascal data structures.

The file *entdir.dat* contains an entry for each entity type, as new types are declared their information is appended to the end of the file.

Each measurement scheme is recorded in the *measdir.dat* file and each conversion scheme in the file *convdir.dat*. Similarly, for every dataset that is declared there is an entry in the *dsdir.dat* file. In each case the new entries are appended to the end of their respective files, when loaded at the start of a run the trees are exactly re-created as they were, which is hopefully in a reasonably balanced form.

The entry in the dsdir.dat file contains the information given in the `ADDDS` command. When the attributes are declared for the dataset, dsname, a file called *<dsname>.att* is generated which will have an entry for each attribute.

Finally, files are created to store the actual data which is entered for the datasets. A separate file is used for each attribute, its name is formed by concatenating the dataset and attribute names, i.e. *<dsname+attname>.dat*. Once the new data has been entered, and possibly sorted into a key order, the files to store the data are completely re-written such that the data can be re-loaded in any key order which exists. This method would be wasteful when adding to large sets of data, but the time taken is not noticable for the size of sets handled in this implementation.

Chapter 6
Statistical Tests and Semantic Requirements

6.1 Background to Statistical Tests

Having consulted a number of statistical packages and textbooks, a selection of measures of association and tests of location (for both 2 and k sample situations) were chosen. In the text of the thesis, the word 'test' is used (for simplicity) to refer to measures of association and tests of location. These tests were selected on the basis of being commonly used and covering a reasonable range of problems for each area.

6.1.1 Measures of Association

(a) Pearson's Product Moment Correlation Coefficient

This is a measure of the strength of a linear relationship between paired samples assumed to be drawn from normally distributed populations.

It is calculated as

$$r = \frac{n\Sigma xy \; - \; \Sigma x \Sigma y}{\sqrt{\left[n\Sigma x^2 - (\Sigma x)^2\right]\left[n\Sigma y^2 - (\Sigma y)^2\right]}} \quad \text{where } -1 \leq r \leq 1$$

The significance of the correlation can be tested using

$$t = r\sqrt{\frac{n - 2}{1 - r^2}} \quad \text{with df} = n - 2.$$

(b) Spearman's Rank Correlation Coefficient

This coefficient measures the strength of a positive or negative relationship between the ranks of two samples. For each pair of ranks the difference ($d_i = X_i - Y_i$) is taken and the coefficient is calculated as

$$r_s = 1 - \frac{6\Sigma d_i^2}{n(n^2 - 1)} \quad \text{where } -1 \le r_s \le 1$$

If the proportion of ties is not too large the effect on r_s is negligible, however for a large number of ties (where t_i is the number of observations tied for rank i) a correction factor can be incorporated so that

$$r_s = \frac{\Sigma x^2 + \Sigma y^2 - \Sigma d^2}{2\sqrt{\Sigma x^2 \, \Sigma y^2}}$$

$$\text{with} \quad \Sigma x^2 = \frac{n(n^2 - 1)}{12} - T_x \quad \text{and} \quad T_x = \frac{\Sigma t_i(t_i^2 - 1)}{12}$$

The significance of the correlation can be tested using

$$t = r_s \sqrt{\frac{n - 2}{1 - r_s^2}} \quad \text{with } df = n - 2$$

(c) Kendall's Rank Correlation Coefficient

Similarly to r_s, this coefficient measures the amount of a positive or negative relationship between two ranked samples, it is calculated as

$$\tau = \frac{2 \ \Sigma a_{ij} b_{ij}}{n(n - 1)} \quad \text{where } -1 \le \tau \le 1$$

$$a_{ij} = \begin{cases} +1 & \text{if } x_i < x_j \\ 0 & \text{if } x_i = x_j \\ -1 & \text{if } x_i > x_j \end{cases} \quad b_{ij} = \begin{cases} +1 & \text{if } y_i < y_j \\ 0 & \text{if } y_i = y_j \\ -1 & \text{if } y_i > y_j \end{cases} \quad \text{for } (i = 1 \text{ to } n\text{-}1,\ j = i\text{+}1 \text{ to } n)$$

In the event of tied ranks (where t_i is the number of ties at rank i) a corrected formula can be used

$$\tau = \frac{\Sigma a_{ij} b_{ij}}{\sqrt{\left(\frac{n(n - 1)}{2} - T_x\right)\left(\frac{n(n - 1)}{2} - T_y\right)}} \quad \text{where } T_x = \frac{\Sigma t_i(t_i - 1)}{2}$$

and similarly for T_y

The significance of τ can be tested using

$$z = \frac{\tau - \mu_\tau}{\sigma_\tau} \quad \text{where } \mu_\tau = 0 \quad \text{and } \sigma_\tau^2 = \frac{2(2n + 5)}{9n(n - 1)}$$

(d) Tau C

This coefficient is derived from Kendall's rank correlation for situations where there are a large number of tied ranks, details of which can be found in Kendall & Stuart (1979). The coefficient is suitable for data arranged as an ordered rxc contingency table and is calculated as

$$t_c = \frac{2m\,\Sigma a_{ij}b_{ij}}{n^2(m-1)} \quad \text{where } m = \min(r, c) \text{ and } -1 \le t_c \le 1$$

The $\sum a_{ij}b_{ij}$ value is computed as in (c) above.

(e) Cramer's V

This measure of association requires an rxc contingency table to be constructed. It is based upon the χ^2 statistic, which can be used to test the significance of the association. The statistic is calculated as

$$V = \frac{\chi^2}{n.m} \quad \text{where } m = \min(r-1, c-1) \text{ and } 0 \le V \le 1$$

(f) Pearson's Coefficient of Contingency

This coefficient is applicable for the same situations as Cramer's V and is also based on a χ^2 statistic calculated from an rxc contingency table.

The coefficient is calculated as

$$C = \sqrt{\frac{\chi^2}{\chi^2 + n}} \quad \text{where } 0 \le C \le \sqrt{\frac{\min(r-1, c-1)}{1 + \min(r-1, c-1)}}$$

6.1.2 Tests for Differences in Two Samples

(a) Normal Statistic

This can be used to compare the means of two samples of size n_1 and n_2. For small samples ($n_1 < 30$ or $n_2 < 30$) it is necessary to assume that the samples are drawn from normally distributed populations with known variances σ_1^2 and σ_2^2. However for large samples ($n_1 \ge 30$ and $n_2 \ge 30$) the central limit theorem applies and no assumptions regarding the population distributions are necessary, in addition if σ_1 and σ_2 are not known then the estimates s_1 and s_2 can be used instead.

The test statistic, which is approximately distributed as a standard normal variable, is calculated as

$$z = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

(b) t Test

The t test is for comparing the means of two samples, for small samples ($n_1 < 30$ or $n_2 < 30$) it must be assumed that the parent populations are normally distributed. There are three forms of the t test which are used in the following situations :-

(i) for independent samples with population variances that can be assumed equal, the test statistic, which exactly follows a t distribution with $df = n_1 + n_2 - 2$, is calculated as

$$t = \frac{\overline{x}_1 - \overline{x}_2}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \quad \text{where } s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

(ii) in the case of independent samples where common population variances cannot be assumed, a statistic which approximates to the t distribution can be calculated, for conservatism the degrees of freedom value is rounded down to the nearest integer

$$t' = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \quad \text{with df} = \frac{\left[\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right]^2}{\frac{\left[\frac{s_1^2}{n_1}\right]^2}{n_1 - 1} + \frac{\left[\frac{s_2^2}{n_2}\right]^2}{n_2 - 1}}$$

(iii) for paired samples, the difference in the pair of values ($d_i = x_{1i} - x_{2i}$) is calculated, no consideration need be given as to whether the population variances are equal since the test statistic uses the differences between the paired observations.

The test statistic is calculated as

$$t = \frac{\bar{d}}{s_d / \sqrt{n}} \quad \text{where } s_d^2 = \frac{\sum d^2 - (\sum d)^2/n}{n - 1}$$

with $n = n_1 = n_2$ and df = n - 1.

(c) Wilcoxon Matched Pairs Signed Ranks Test

This test can be used to detect any kind of difference in two paired samples (with the null hypothesis being that there is no difference). The absolute differences between each matched pair are obtained and ranked in order of magnitude. The sums of the +ve and -ve differences are then calculated, which under the null hypothesis would be about equal. Where there is no difference in a pair of observations, i.e. d=0, the pair is dropped and the value of n (the number of pairs) is reduced accordingly. For pairs with the same absolute difference average ranks are assigned.

Let T equal the smaller of the two sums of ranks. For small values of n, tables are consulted to test the significance of the T value. For larger samples (typically $n > 25$) the value of T is approximately normally distributed with

$$\mu_T = \frac{n(n+1)}{4} \quad \text{and } \sigma_T^2 = \frac{n(n+1)(2n+1)}{24}$$

If there are tied ranks, a corrected form of σ_T^2 is calculated as

$$\sigma_T^2 = \frac{n(n+1)(2n+1) - {}^1/_2 \sum_{j=1}^{g} t_j(t_j - 1)(t_j + 1)}{24}$$

where g is the number of tied groups and t_j the number of observations tied for rank j.

The significance of T is then tested using the standard normally distributed variable

$$z = \frac{T - \mu_T}{\sigma_T}$$

(d) Sign Test

This test is appropriate for establishing a contrast in two related samples and requires only that a difference, +ve or -ve, can be found for each pair of observations.

Let x be the lower of the sums of +ve and -ve differences, tied observations (where there is no difference in the pair of values) are dropped from the data and the value of n reduced. The null hypothesis which is tested is that the median difference is zero. For small samples (typically $n \leq 25$) the significance of x is tested by consulting cumulative binomial tables. For larger values of n, x is approximately normally distributed with

$$\mu_x = \frac{n}{2} \quad \text{and } \sigma_x = \frac{\sqrt{n}}{2}$$

The significance of x is then tested using

$$z = \frac{x - \mu_x}{\sigma_x}$$

The test actually assumes that the data comes from an underlying continuous distribution, a correction term for continuity can be incorporated giving a test statistic of

$$z = \frac{(x \pm 0.5) - \mu_x}{\sigma_x} \quad \text{use} \begin{cases} x + 0.5 \text{ when } x < n/2 \\ x - 0.5 \text{ when } x > n/2 \end{cases}$$

(e) Mann-Whitney U Test

This test is a nonparametric alternative to the t test and determines whether two independent samples have been drawn from the same population.

Let n_1 = the number of cases in the smaller of the two groups,

n_2 = the number of cases in the larger of the two groups.

The samples are combined and ranked where

R_1 = the sum of the ranks assigned to the group of size n_1,

R_2 = the sum of the ranks assigned to the group of size n_2.

The value of U is taken as the smaller of the two values U' and U" which are calculated as

$$U' = n_1 n_2 + \frac{n_1(n_1 + 1)}{2} - R_1 \quad \text{and} \quad U'' = n_1 n_2 + \frac{n_2(n_2 + 1)}{2} - R_2$$

For small samples, the significance of U is tested by consulting tables of critical values. For larger samples

$$\mu_u = \frac{n_1 n_2}{2} \quad \text{and} \quad \sigma_u^2 = \frac{n_1 n_2 (n_1 + n_2 + 1)}{12}$$

In the event of ties average ranks are assigned. If the ties are just between observations in the same group there is no effect, however if ties occur between groups (where t_i is the number of observations tied for rank i) then a corrected form of σ_u is used.

$$\sigma_u^2 = \left[\frac{n_1 n_2}{N(N-1)} \right] \left[\frac{N(N^2 - 1) - \sum(t_i^3 - t_i)}{12} \right] \quad \text{where } N = n_1 + n_2$$

The significance of U is tested using the standard normally distributed variable

$$z = \frac{U - \mu_u}{\sigma_u}$$

(f) McNemar Test

This test is suitable for paired samples which have been represented on a dichotomy. A contingency table can be constructed to summarise the data.

		SAMPLE Y	
		0	1
SAMPLE X	0	A	B
	1	C	D

The test ignores the observations for pairs which have the same value, i.e. cells A and D, but concentrates on those pairs where the values are different. The test examines whether or not there is a difference in the probability of one combination of values (0,1) against the other (1,0). The test statistic, shown below, is calculated to compare the observed with the expected distribution of observations in cells B and C.

$$\chi^2 = \frac{[B - (B + C)/2]^2}{(B + C)/2} + \frac{[C - (B + C)/2]^2}{(B + C)/2} = \frac{(B - C)^2}{B + C}$$

Since the chi-squared distribution, which is continuous, is used to approximate a discrete distribution a correction for continuity can be included so that the test statistic is calculated as

$$\chi^2 = \frac{(|B - C| - 1)^2}{B + C} \quad \text{with } df = 1.$$

(g) Fisher Exact Probability Test

This test is applicable for data from two independent samples which fall into a dichotomy, as shown below.

	CLASS		
	0	1	
SAMPLE X	A	B	A+B
SAMPLE Y	C	D	C+D
	A+C	B+D	N

The test compares the two samples to see if the data differs in the proportions with which it falls between the two classes. Given that the marginal totals are fixed, the probability of the observed distribution of values can be calculated as

$$p = \frac{(A + B)!\ (C + D)!\ (A + C)!\ (B + D)!}{N!\ A!\ B!\ C!\ D!}$$

By also calculating the probabilities of the more extreme deviations, and summing these probabilities, the null hypothesis can be tested. For a two-tailed test the probabilities are doubled.

6.1.3 Tests for Differences in K Samples

(a) Randomised Block Design

A randomised block design can be used to test for differences in population means of k related samples. It is assumed that the observations are drawn from normally distributed populations which have a common variance. The data is analysed as a two-way ANOVA without interaction, with one factor assigned to samples (k levels) and the other to blocks (n levels). The model is

$$y_{ij} = \mu + \beta_i + \tau_j + e_{ij} \quad (i = 1 \text{ to } n, j = 1 \text{ to } k)$$

(b) One-Way Analysis of Variance

A one-way ANOVA can be used to detect differences in the population means of k independent samples, each of size n_i. The analysis assumes that the populations are approximately normal with a common variance. The one-factor model that is used is

$$y_{ij} = \mu + \beta_i + e_{ij} \quad (i = 1 \text{ to } k, j = 1 \text{ to } n_i)$$

(c) Friedman Two-Way ANOVA by Ranks

The Friedman test is a nonparametric alternative to the randomised block design and tests if k related samples are drawn from the same population. The data is considered as being arranged in a table with

N rows	- the number of observations for each sample,
K columns	- the number of samples.

Each row is ranked from 1 to k, with tied values being assigned average ranks. The sum of ranks (R_i) is calculated for each column, which under the null hypothesis would be about equal.

The test statistic is calculated as

$$Q = \frac{12}{NK(K+1)} \sum_{i=1}^{k} R_i^2 - 3N(K+1)$$

In the event of tied values average ranks are assigned, a corrected form of the test statistic can be calculated, where t_{jk} is the number of ties for row j with rank k.

$$Q = \frac{\frac{12}{NK(K+1)} \sum_{i=1}^{k} R_i^2 - 3N(K+1)}{1 - \frac{\Sigma t_{jk}^2(t_{jk} - 1)}{NK(K^2 - 1)}}$$

When the number of rows and columns is not too small the statistic follows a χ^2 distribution with df = K - 1, otherwise a table of critical values must be consulted to obtain the significance.

(d) Kruskal-Wallis One-Way ANOVA

The Kruskal-Wallis test is a k sample generalisation of the Mann-Whitney U test and is useful for detecting whether k independent samples are drawn from different populations.

The samples are combined and ranked where

n_j = the number of cases in the jth sample ($\Sigma n_j = N$),

R_j = the sum of ranks for the jth sample.

The test statistic is calculated as

$$H = \frac{12}{N(N+1)} \sum_{j=1}^{k} \frac{R_j^2}{n_j} - 3(N+1)$$

In the event of tied values, where t_i is the number of ties for rank i, a corrected form of the test statistic is calculated as

$$H = \frac{\frac{12}{N(N+1)} \sum_{j=1}^{k} \frac{R_j^2}{n_j} - 3(N+1)}{1 - \frac{\sum t_i(t_i^2 - 1)}{N(N^2 - 1)}}$$

The value of H is approximately distributed as a χ^2 distribution with df = k - 1.

(e) Cochran Q Test

This test is applicable for k related samples with dichotomised data. The test establishes whether the proportion of responses for each category is the same for each sample. A two-way table with n rows and k columns is constructed, consider the categories of the dichotomy to be types A and B.

Let G_j = the total number of A responses in column j,

L_i = the total number of A responses in row i.

The test statistic, which is approximately a χ^2 distributed variable with df = k - 1, is calculated as

$$Q = \frac{(k-1)\left[k\sum_{j=1}^{k}G_j^2 - (\sum_{j=1}^{k}G_j)^2\right]}{k\sum_{i=1}^{n}L_i - \sum_{i=1}^{n}L_i^2}$$

(f) Chi-Squared Test

A chi-squared test can be used for k independent samples, measured as discrete categories. It detects any difference in the distribution across the categories among the samples where

$$\chi^2 = \sum_{i=1}^{k}\sum_{j=1}^{n}\frac{(O_{ij} - E_{ij})^2}{E_{ij}} \quad \text{with df} = (k-1)(n-1)$$

The size of the expected frequencies should be checked, it is common to state that it is desirable if fewer than 20% are less than 5 and none are less than 1.

6.2 Semantic Requirements Overview

The objective of the system is to try and ensure that statistical tests are used which are appropriate for the data concerned. Siegel (1956) noted that "associated with every statistical test is a model and a measurement requirement; the test is valid under certain conditions, and the model and measurement requirement specify those conditions". The statistical model was identified as being the nature of the population and the manner of the sampling. The user will request that a statistical test is to be performed on a number of samples, each specified using a format of <dsname>.<attname>, the system then performs a number of checks which use the information in the semantic data model and the actual data values to verify some of the model and measurement requirements.

Nelder (1977) had discussed three alternatives for checking the assumptions of a statistical test :-

(i) for the system to apply a number of standard checks by default;

(ii) for a number of standard checks to be provided for the user to request;

(iii) for a set of low level functions to be provided, a tool-kit, for the user to program their own checks.

The needs of the inexperienced user are best met by the first alternative and this was the method adopted. Although the checks are applied by default, the user does have some control as the results are not all automatically accepted by default. It is extremely difficult, if not impossible, to specify a set of hard and fast rules which could be applied to automatically decide if the use of a test was sound. Depending upon the particular circumstances, some of the requirements need not be exactly met, for example the assumption that data is normally distributed. In the case of subjective issues, the system permits a user's wish to prevail. This provides a further argument against adopting an authoritarian approach. The implementation aspects of this are discussed in the next chapter.

The checks that have been implemented are performed at two levels, they validate the following :-

(i) that a class of tests is applicable, e.g. measures of association;

(ii) that a specific test is the appropriate type of test, e.g. Pearson's PMCC.

The initial objective was that given a number of arguments by the user, the appropriate type of test checks would be applied, if successful the system would then apply the checks for the particular statistical test specified. It seemed desirable that if the type of test was applicable, but the test chosen by the user was not appropriate, for the system to be able to recommend one that was. A natural extension to this was to allow a user to just specify the type of test required, e.g. measure of association, if the type of test is applicable the system would recommend a particular test to the user. The user can therefore either enter the name of a specific test if one is known, or if not just the name of the type of test required. The system performs the class of test checks on the arguments given and forms them into groups that the type of test is applicable to. If the user has given suitable arguments then there will be just the one group. The checks for a particular test can then be applied to each of those groups of arguments. Allowing the program to operate in this manner has been made easier because of the hierarchical organisation of the semantic requirements of the tests.

In recommending a measure of association, the system will choose a test to make the best use of the level of measurement of the data. All of the tests of location have been grouped together and the strategy adopted in selecting one is to: make the best use of the level of measurement; take advantage if the samples are paired or related; select a test specifically for two samples rather than a general test for k samples. Should the user specify a test that does not make the best use of the data the system will still allow its use. An alternative, that has not been implemented, would be to allow the use of the test but to point out to the user that a more powerful test may be applicable.

6.3 Representation of Semantic Requirements

The semantic requirements of the statistical tests must be represented in a manner that permits the two modes of use envisaged, those being :-

(i) to look at the requirements of a user specified test to see if it is appropriate;

(ii) to recommend a test suitable for the arguments given, this involves looking at the requirements to identify a test.

This desired objective has been achieved by using a representation as depicted in Figure 6.1. Three arrays of pointers have been used to represent the following sets of requirements :-

(i) *class_checks* - indexed by the name of the type of test, i.e. association and location;

(ii) *assoc_checks* - indexed by the name of the measure of association;

(iii) *loc_checks* - indexed by the name of the test of location.

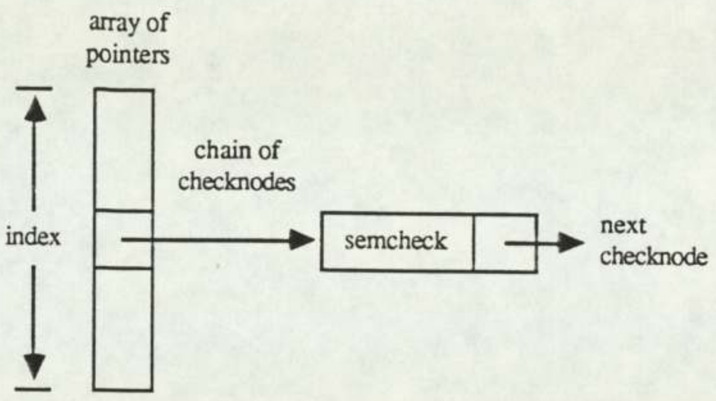

Figure 6.1 Semantic Requirements Representation

The pointers in the arrays reference a chain of checknodes. Each checknode contains a *semcheck* field - this is a keyword identifying a semantic requirement - and a pointer to the next node in the chain. This arrangement allows a varying number of requirements to be specified.

To verify the requirements of a type of test the index of the class_checks array is used to locate the appropriate pointer. Each node in the chain is processed one at a time with the keyword identifying the requirement that is to be validated. The corresponding array of test requirements, assoc_checks or loc_checks, is then used to perform the second stage of checking.

If the user has specified a particular test, the relevant chain of checknodes is located using the index of test names. The requirements are considered in the order in which they occur in the chain. If a requirement is not satisfied the test is deemed inappropriate and any remaining requirements are ignored. If however the end of the chain is reached then the test is accepted as being applicable.

When the system is attempting to select a test to recommend to the user, either because only the type of test was given or if the test preferred by the user was inappropriate, the system must consider the requirements to identify a test name. For the assoc_checks and loc_checks arrays the indices are ordered according to the strategy for choosing a test. For example, a two sample test for paired interval level

6.1a Classes of Test

Association	relatedinst	
Location	simenttype	eqdomains

6.1b Measures of Association

Pearson		intqnt	normal
Spearman		ranked	
Kendall		ranked	
Tau_c		ordqlt	
Cramers_V		nomcat	chifreq
Coeff_of_cont		nomcat	chifreq

6.1c Tests of Location

Normal_test	twosample		eqintqnt		nige30
T_paired	twosample	relatedinst	eqintqnt	normal	
Randomised_block		relatedinst	eqintqnt	normal	eqvar
T_common	twosample		eqintqnt	normal	eqvar
T_separate	twosample		eqintqnt	normal	
One_way_AOV			eqintqnt	normal	eqvar
Wilcoxon	twosample	relatedinst	eqordqnt		
Sign_test	twosample	relatedinst	eqordqlt		
Friedman_AOV		relatedinst	eqordqlt		
Mann_Whitney	twosample		eqordqlt		
Kruskal_Wallis			eqordqlt		
McNemar_test	twosample	relatedinst	eqdichcat		
Cochran_Q		relatedinst	eqdichcat		
Chi_squared			eqnomcat		chifreq
Fisher_exact	twosample		eqdichcat		

Table 6.1 Requirements of Statistical Tests

data appears before a similar test that does not assume paired samples. The tests can therefore be considered in index order, the first one for which all of the requirements are satisfied will be the one recommended.

Table 6.1 shows the requirements that have been specified for the type of tests and the specific tests, the meanings of which are described in section 6.4. The requirements are ordered in the chains from left to right as they appear in the table. They are ordered such that they can be sensibly applied, that is there is no point in checking for normality until it is known that the data is quantitative.

The requirements for the three arrays are stored in the files *class_checks.dat*, *assoc_checks.dat* and *loc_checks.dat*, the information is read into the Pascal data structures when the program in invoked.

6.4 Description of Semantic Requirements

The requirements keywords that can be specified are described in the following sections, some of which study each sample in isolation to see if a condition is satisfied whereas others will consider the group of samples as a whole to examine the existence of a required common characteristic.

6.4.1 Homogeneous Entities

It may be required that for a sensible application of a test the data should be measured for similar types of objects. That is the datasets of the attributes that are involved should be measuring data for entities that are of a similar type, this requirement is identified by the keyword *simenttype*. It can be validated by searching the entity type taxonomy. From the entity types referenced in the dataset nodes it is possible to work up the entity type nodes in the taxonomy, whilst the links are denoting a generic relationship, to find the most generic entity types for each of the datasets. The requirement is satisfied if the most generic nodes identified in the taxonomy for each of the datasets are the same.

6.4.2 Related Samples

A number of the tests are intended for use with paired or related samples, specified with the keyword *relatedinst*. That is the instances of the datasets for each of the attributes that have been given should somehow match up. For samples that are attributes from the same dataset no further effort is required. However if the datasets for the attributes are different then each dataset must have a key field where the values of the attributes comprising the key coincide. Since the instances of a dataset are stored in key order it is simple to see if two datasets have the same set of key values.

6.4.3 Number of Samples

Some tests are restricted to being used specifically in a *twosample* situation whereas there are others that are more general and are applicable for k samples (*ksample*), where $k \geq 2$.

6.4.4 Homogeneous Properties

The objective of performing a test of location is often to decide whether or not a number of samples could have been drawn from the same or similar populations. This is achieved by examining the distribution of the sample values by means of some parametric or nonparametric technique. For this to be a meaningful operation the data under consideration should be comparable, that is like should be compared with like. In this situation the requirement *eqdomains* is specified. The att_type field of an attribute node denotes what property is being represented. For a comparison of data values to be appropriate this field should have the same value for each of the attributes, in this way the system can ensure that heights are not compared with weights.

As well as the att_type value the method of recording the data must also be taken into consideration, this aspect of homogeneity is covered in the next section.

6.4.5 Measurement

It is important to take account of the measurement of the data in deciding the soundness of applying a statistical test, since all tests assume something of the data. The measurement requirements are concerned with both the level of measurement and the measurement scheme used to record the data.

The vast majority of textbooks on statistical techniques use the level of measurement as the basis for deciding upon the type of data for which a particular test is applicable. It is in fact common for such textbooks to use the assumed level of measurement of the data as a means of classifying the tests. An alternative approach that has been adopted by some, notably Marascuilo & McSweeney (1977), is to concentrate on stipulating these measurement conditions with respect to the distributional properties of the data. Statistical data is identified as being one of two types :-

(i) qualitative - which is subdivided into ordered and unordered data;

(ii) quantitative - which has the subclasses discrete and continuous.

The requirement for the correct application of a test concerned with the measurement aspect of the data can then be specified with respect to the four classes of data identified.

For the most part the level of measurement is used as the foundation upon which the various semantic measurement requirements are specified.

With regards to parametric statistical tests, some require the data to have a ratio level of measurement but for most of them data which is at least interval will suffice.

Amongst the most powerful of the nonparametric tests are those which are based on ranking the data. Such tests can be used for interval and ratio level data where all of the conditions for the use of a parametric test are not satisfied, for example if it cannot be assumed that the data is drawn from a normally distributed population. Alternatively, it is generally accepted that most rank tests are applicable to ordinal level data, with corrected formulae being used in the event of tied ranks. Earlier publications suggested that the data should have an underlying continuous distribution but more recently it has been felt that this assumption is unnecessarily restrictive. Conover (1980) advocated that most rank tests were suitable as long as the sample values can take more than one possible value, i.e. $P(X = x) < 1$ for each x, the theory underlying this belief appears in Conover (1973). This standpoint has been adopted and the system will allow most rank tests to be used with ordinal level data.

One test where there appears to be less of a consesus of agreement is the Wilcoxon Matched Pairs Signed Ranks test, this ranks the absolute differences of each pair of values. Siegel (1956) considers that the differences should be at least ordinal, a footnote indicates that this really requires the data to lie at least between an ordinal and interval scale. Lehmann (1975) does not mention the level of measurement but states that it is desirable to avoid ties whereas Marascuilo & McSweeney (1977) classify the test as being suitable for quantitative data. Yet another alternative is given by Conover (1980) who regards the test as being for interval level data. It was decided that for this test the requirement would be for the data to be quantitative, which could be ratio, interval or ordinal. Most of the sources of reference given seem to regard the number of ties to be the crucial point and requiring quantitative data will often keep the number of ties down, although this cannot of course be guaranteed. An alternative course of action would be to also allow ordered qualitative data where the number of ties is not too many. Since it was not possible to find an agreed quantified value for 'not too many' this idea was discarded.

Rank tests have been discussed in relation to ratio, interval and ordinal level data. For tests which rank the samples individually, rather than in some collective way, a further more relaxed requirement allows attributes with a *datalevel* value of ratio, interval, ordinal or rank to be used.

Other nonparametric tests, often based on contingency tables, require only that the data can be grouped into ordered or unordered categories. Some tests are more specific and it is necessary for the categories to be a dichotomy.

For tests based upon the assumption of homogeneous properties an additional measurement requirement is that the measurement scheme used for each sample should be the same. The measurement level and measurement scheme requirements are considered together and given as a single keyword, there is one specified for each test and their requirements are summarised as follows :-

(i) *eqratqnt* - ratio level data that can be represented with the same measurement scheme;

(ii) *ratqnt* - ratio level data;

(iii) *eqintqnt* - interval or ratio level data that can be represented with the same measurement scheme;

(iv) *intqnt* - interval or ratio level data;

(v) *ranked* - ordinal, rank, interval or ratio level data;

(vi) *eqordqnt* - ordinal, interval or ratio level data that can be represented with the same quantitative measurement scheme;

(vii) *eqordqlt* - ordinal, interval or ratio level data that can be represented with the same measurement scheme;

(viii) *ordqlt* - ordinal, interval or ratio level data;

(ix) *eqnomcat* - data that can be represented with the same qualitative measurement scheme;

(x) *nomcat* - data that can be represented with a qualitative measurement scheme;

(xi) *eqdichcat* - data that can be represented with the same dichotomous qualitative measurement scheme.

The system will initially check the attribute node for each argument to see that the level of measurement values satisfy the requirement. If these are suitable the system may then need to examine the measurement schemes to see if the data is in an appropriate form for the test. To obtain the form desired it may be necessary to convert the data for some or all of the attributes involved to another measurement scheme. The need to perform a data conversion will be due to one or a combination of the following reasons :-

(i) to get all of the data recorded using the same measurement scheme;

(ii) to categorise quantitative data;

(iii) to form the data into a dichotomy.

6.4.6 Normality

A number of the statistical tests assume that the samples are drawn from normally distributed populations, an assumption specified using the keyword *normal*. The requirement is satisfied for a sample if the *att_dist* field of the attribute node has the value *normaldist* or the number of instances is 30 or more, since in the latter case the central limit theorem applies. Otherwise a test is applied to determine the likelihood of normality, although such tests cannot guarantee complete accuracy they can act as a guide. The test chosen was the one presented by Shapiro & Wilk (1965) which is applicable on a single sample, there may be a more recent test which has more desirable characteristics and if so it would be simple to do a substitution.

The test is applied to a random sample of size n, $x_1,x_2,...,x_n$, which is ordered such that $y_1 \leq y_2 \leq ... \leq y_n$.

The values

$$b = \sum_{i=1}^{k} a_{n-i+1}(y_{n-i+1} - y_i) \qquad \text{where} \quad k = n \text{ DIV } 2$$

$$\text{and} \qquad S^2 = \sum_{i=1}^{n} (y_i - \overline{y})^2$$

are computed using the tabulated values of a_{n-i+1}.

The test statistic is then calculated as

$$W = \frac{b^2}{S^2}$$

and the significance evaluated using tables of critical values of W.

The file *shapwilkcoeff.dat* contains the values of a_{n-i+1} in ascending order of i from 3 to 29 followed by the 5% critical values of W for the corresponding values of n. These values are read from the file into Pascal arrays which are then used by the routine.

6.4.7 Equality of Variances

A requirement that is specified for some tests is that the samples should be drawn from populations having equal variances, denoted by the requirement keyword *eqvar*. In a two sample situation the F test is used with the test statistic computed as

$$F = \frac{s_1^2}{s_2^2}$$

To test the significance of the statistic, use is made of a NAG library routine to return the probability associated with the calculated F value.

For the more general k sample situation a test presented by Bartlett (1937) is used. This examines the equality of k normally distributed samples, with the ith sample having n_i elements and $\sum n_i = N$. The sample variances (s_i^2) are calculated with each having $\phi_i = n_i - 1$ degrees of freedom.

The average of the estimated variances is calculated as

$$s^2 = \frac{\sum s_i^2 \phi_i}{\Phi} \quad \text{where } \Phi = \sum \phi_i$$

It is then possible to compute

$$M = \Phi \ln(s^2) - \sum_{i=1}^{k} \phi_i \ln(s_i^2) \quad \text{and} \quad A = \frac{1}{3(k-1)}\left[\sum_{i=1}^{k} \frac{1}{\phi_i} - \frac{1}{\Phi}\right]$$

with the the test statistic being

$$\frac{M}{1 + A}$$

As noted, this statistic assumes that the samples are normally distributed, if this assumption cannot be made a modification according to Box (1953) is used with the test statistic

$$\frac{M}{1 + \frac{\gamma_2}{2}} \quad \text{where} \quad \gamma_2 = \beta_2 - 3$$

An estimate given by Anscombe (1960) is used for γ_2

$$\hat{\gamma}_2 = \frac{N^3}{\upsilon(\upsilon+2)\ \{1+(N-1)\rho^4\}\ -\ 3N}\left[\frac{\upsilon+2}{\upsilon}\ \frac{\Sigma e_{ij}^4}{(\Sigma e_{ij}^2)^2}\ -\ \frac{3}{N}\right]$$

$$\text{where} \quad \upsilon = N - k\ , \quad \rho^2 = \frac{k}{\upsilon(N - 1)} \quad \text{and} \quad e_{ij} = x_{ij} - \overline{x}_i$$

Both Box's and Bartlett's test statistics are distributed as χ^2 variables with k-1 degrees of freedom, a call to a NAG library routine returns the probability associated with the computed statistic.

As with testing for normality, different or more extensive statistics may be more appropriate to compare the equality of variances and the implementation could be easily changed to accommodate them.

6.4.8 Size of Samples

The normal test is often recommended for comparing the means of two large independent samples. For large samples, where $n_i \geq 30$ is commonly taken as being large, the central limit theorem means that no assumptions regarding the population distributions need be made and in addition the population variances need not be known since the sample estimates can be used instead. It is usual for the t test to be recommended in the case of small samples. To accommodate this convention the requirement *nige30* is specified for the normal test. Each sample is checked to see that 30 or more instances have been declared for the datasets of the attributes involved.

6.4.9 Expected Frequencies in Contingency Tables

A number of tests for categorised data involve the calculation of the χ^2 statistic from a contingency table. The existence of small values of expected frequencies can result in a large distortion occuring in the test statistic. Textbooks recommend the pooling of categories to avoid these undesirable small frequencies. The definition of small is not however consistent. Some recommend that no expected frequencies should be less than 5 whereas others consider that none less than 1 is more reasonable. A common condition that is given is that none should be less than 1 and only 20% less then 5. This last alternative has been adopted and is specified with the *chifreq* keyword.

6.5 Résumé

In identifying a set of requirements that can be specified for a type of test or a specific test to be applicable, use has been made of all of the semantic knowledge that is represented in the data model. The keywords simenttype, relatedinst, eqdomains, eqratqnt, ratqnt, eqintqnt, intqnt, ranked, eqordqnt, eqordqlt, ordqlt, eqnomcat, nomcat and eqdichcat are concerned with the knowledge about objects, instances, properties and measurement. In addition, there are others which can be regarded as being more involved with numeric issues and the actual data, those being normal, eqvar, chifreq, twosample, ksample and nige30.

Chapter 7

The Operation of Validating the Use of a Statistical Test

7.1 Preliminaries

A request to perform a statistical test can be made by entering either the name of a class of tests or the name of a specific test. This request is accompanied by the list of arguments that the test is to be applied to, that is

```
either      <testclass>  <dsname>.<attname> <dsname>.<attname> ...
or          <testname>   <dsname>.<attname> <dsname>.<attname> ...
```

The function of the main program is to read in the name of a command entered by the user and to call the relevant procedure, which will then read in any arguments and process the command. For a request to perform a statistical test the procedure *procstatreq* is called, the actual parameters passed to it are dependent upon the command entered, the name of the command is parsed and identified by the variable token.ttype. The values of the parameters passed to procstatreq are determined by the following section of code from the main program.

```
CASE token.ttype OF
 ...
 ...
 association : procstatreq(association, nulltest, twosample);
 location    : procstatreq(location, nulltest, ksample);
 pearson..coeff_of_cont :
               procstatreq(association, token.ttype, twosample);
 normal_test..fisher_exact :
               procstatreq(location, token.ttype, ksample);
 ...
 ...
END;
```

The first parameter of the procedure procstatreq identifies the type of test required, this information is inferred if the name of a specific test is entered. The second parameter denotes whether the user has requested any specific test, a value of *nulltest* indicates that only the desired type of test name was given. The final parameter signifies the number of arguments that can be given for the type of test. This requirement is given as a parameter rather than as a semantic requirement in the class_checks array since it serves a number of functions. It is used to check both the syntax and the semantics of the original command as it is entered and is used again once the class of test checks have been performed. These operations are explained in later sections of the chapter.

The first task of the procedure procstatreq is to read in the arguments that the user has supplied. As was briefly mentioned in section 6.2, the objective of performing the checks at the type of test level is to form these arguments into groups such that a test of the required type can be applied to each group. The information about the arguments must be represented in a form that allows the organisation of the groups to be depicted. A representation as shown in Figure 7.1 has been used to facilitate this need.

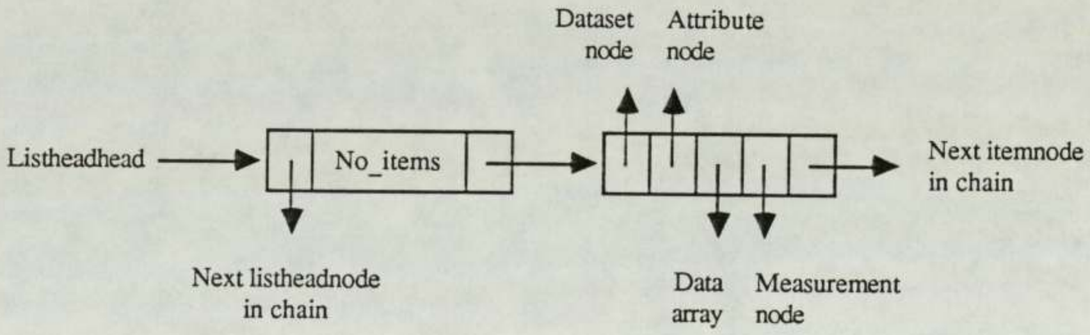

Figure 7.1 Argument List Representation

The variable *listheadhead* points to the head of a chain of *listheadnodes*, there is one such node for each list (or group) of arguments. The information about an argument in a list is represented in an *itemnode*, for each listheadnode there will be a chain of itemnodes, one for each argument in the list.

A listheadnode contains the following fields :-

(i) nexthead - a pointer to the next listheadnode in the chain;

(ii) no_items - an integer recording the number of nodes in the chain of itemnodes, i.e. the number of arguments in the list;

(iii) itemhead - a pointer to the head of the chain of itemnodes.

The fields of an itemnode represent the information about an argument as follows :-

(i) dsinfo - a pointer to the dataset node of the argument;

(ii) attinfo - a pointer to the attribute node of the argument;

(iii) convdata - this field is a pointer to a data array and will initially have the value NIL, if it is necessary to convert the original data (referenced by a pointer in the attribute node) to satisfy a requirement, an array referenced by this pointer will be generated to contain the converted data;

(iv) measinfo - a pointer to an entry in the measurement directory identifying the measurement scheme of the converted data, this pointer will have the value NIL until a conversion is made;

(v) nextitem - a pointer to the next itemnode in the chain.

To satisfy a particular measurement requirement it may be necessary to convert the data for some or all of the arguments to a different measurement scheme. This requires an extra set of data to be generated, since the original copy must be left as it is. By using a field in the itemnode to reference the converted data, the system can easily check which arguments required conversions to be made. In addition, since the generated data is only required for the current command, it can easily be disposed of once the system has completed processing the steps of the command. If a test were to be applied, any converted data referenced in the itemnode would be used instead of the original data located via the attribute node.

When the procedure procstatreq is invoked a listheadnode is generated and initialised, to begin with there will be just one list of arguments. As the information about each argument is read in and validated, an itemnode is generated and added to the chain, the no_items field in the listheadnode is also incremented. For an argument to be valid at least 3 instances must have been declared for the dataset, i.e. the attribute will have 3 or more items of data.

Once the end of the list of arguments has been reached, the system uses the no_items value in the listheadnode to check that the number of arguments entered is valid for the type of test required. In the event of an error occuring, either in validating a particular argument or checking the number of arguments, an error message is output and the command aborted. In this situation the listheadnode and any itemnodes that have been generated are disposed of. If however the arguments of the command have been parsed and no error has resulted, the first stage of applying the semantic requirements, at the class of test level, is initiated.

7.2 The Class of Test Level Operation

The actions performed at the class of test level are divided into two stages :-

(i) applying the semantic requirements and if necessary splitting the arguments into groups;

(ii) reviewing the resultant organisation of the arguments to see if the system can proceed to the test level stage and if needed reporting the outcome back to the user.

The controlling procedure procstatreq calls the procedures *checkclassreq* and *reviewclasschecks* for the former and latter tasks respectively.

7.3 Applying the Class of Test Requirements

The procedure checkclassreq is called from procstatreq as

checkclassreq(class_checks[testclass])

The actual parameter is a pointer to the head of the chain of checknodes identifying the requirements of the class of tests required. The overall operation of the procedure can be seen as

```
for each semantic requirement
        consider each group in turn and divide into any subgroups
```

For example, in the case of a test of location there are two semantic requirements that must be processed, the above operation will therefore be as follows :-

(i) the original single list of arguments is considered and formed into groups of those having a similar entity type;

(ii) each of the groups formed after (i) are considered independently and those arguments measuring the same quality or quantity are formed into subgroups.

After processing the second semantic requirement each listheadnode will reference a chain of itemnodes, each denoting arguments having data measured for similar entity types *and* measuring the same quality or quantity.

This exercise may involve manipulating the argument list representation to remove an itemnode from one list and either adding it to another existing list or creating a new one. An example of how this manipulation is achieved is illustrated in Figure 7.2. Part of an argument list representation is shown in Figure 7.2A, one or more semantic checks have been applied and the arguments have been split into a number of groups. A semantic requirement is about to be applied to the group of arguments in list_1, the current list of interest being identified by the pointer variable *listtocheck*. The pointer *lastlisthead* will be used to indicate the last listheadnode that has been created for itemnodes that are currently in listtocheck. Lastlisthead is therefore initially set to listtocheck since no extra listheadnodes have yet been created. The system decides if an item is appropriate for the list it is currently in by comparing it with the item at the head of the list of itemnodes. This comparison is done according to the criteria of the semantic requirement. If the criteria is not met then the itemnode must be removed from the list. It can then be compared with those items at the head of any lists up to and including that identified by lastlisthead and added to a list if appropriate. If it is not compatible with any of these then a new listheadnode is created for it, this node is placed after the one referenced by the pointer lastlisthead and the pointer is then updated. For the situation depicted in Figure 7.2A, itemB is compared with itemA and the requirement is for example not met. Since there are no other lists that itemB can be added to a new listheadnode, list_1a, is created and the pointer lastlisthead advanced. The new state of that part of the argument list representation is shown in Figure 7.2B. ItemC will then be compared with itemA to ascertain as to whether it should stay in

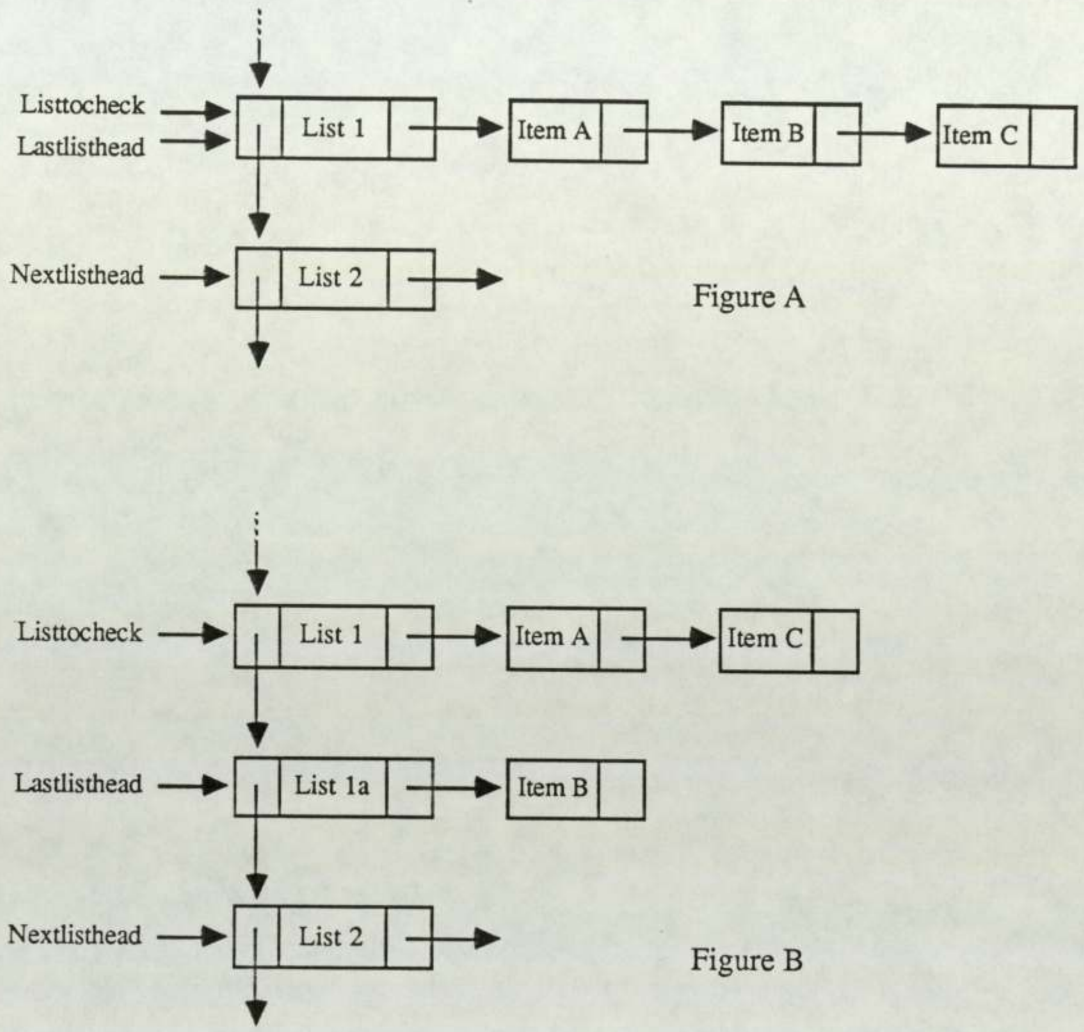

Figure 7.2 Manipulation of Argument List Representation

list_1, if not it would be compared with itemB and possibly added to list_1a or may require a new listheadnode to be created. Once the itemnode at the tail of listtocheck has been processed, those arguments represented in lists up to and including that identified by lastlisthead will have been grouped according to the requirements up to the current one being applied. The listheadnode referenced by the pointer *nextlisthead* then becomes the next list to check.

Having applied all of the semantic requirements for the class of test, the system will then report back any changes in the organisation of the argument list representation to the user, as described in the next section.

7.4 Reviewing the Results of the Class of Test Requirements

The procedure reviewclasschecks is called from procstatreq as

reviewclasschecks(testclass, typeoftestargs)

The actual parameters that are passed enable the system to identify :-

(i) the class of tests required, and hence the semantic requirements that have been applied to the argument list from the class_checks array;

(ii) the number of arguments that must be represented in each list for it to be possible to apply a test, i.e. twosample or ksample.

Having applied the class of test level semantic requirements to the initial single list of arguments, the first task of the procedure is to determine if the resulting groups of arguments, identified by the existence of a listheadnode in the chain referenced by the pointer variable listheadhead, contain sufficient members for it to be feasible to perform a test. The no_items field of each listheadnode is inspected to see if it conforms to the range required by the typeoftestargs parameter. Those listheadnodes containing sufficient itemnodes for a test to be applied are chained together and referenced by the pointer variable *validlists*. Conversely, the pointer variable *invalidlists* identifies a chain of any listheadnodes representing groups with too few members.

Once all of the listheadnodes have been assigned to either validlists or invalidlists, the action of the procedure will be dependent upon which of the three possible situations has occured, those being that :-

(i) the validlists chain is empty.

No test can be applied since none of the groups formed as a result of applying the class of test requirements contained enough arguments, the user is informed that the arguments are such that no test of the desired class is applicable.

(ii) there is at least one listheadnode in the invalidlists chain or one or more listheadnodes in the validlists chain.

That is, after the class of test level of checking it has been found that it is not possible to perform a single test on all of the arguments.

If the invalidlists pointer is not NIL then the arguments represented by the chain of listheadnodes are displayed and the user told that a test of the class specified cannot be applied to any of those arguments.

The system then informs the user of those arguments that can be used. That is, if validlists contains one listheadnode then a single test can be applied whereas if there is more than one listheadnode in the chain a test can be applied to each of the groups of arguments.

For example, if seven arguments were entered the groups could be in a form that results in the following output.

```
Cannot apply a test of location to the following argument(s)

<dsname>.<attname>
<dsname>.<attname>

Can apply a test of location to each of the following groups

<dsname>.<attname>
<dsname>.<attname>

<dsname>.<attname>
<dsname>.<attname>
<dsname>.<attname>
```

The user is then asked if they wish the system to continue and attempt to apply a test to those arguments where possible. If the user decides that there is no benefit in continuing, either because of the arguments that cannot be used or due to the grouping of the arguments, the listheadnodes and itemnodes referenced by the pointer validlists are disposed of and the pointer is set to NIL.

(iii) there is one listheadnode in the validlists chain and the invalidlists chain is empty.

In this situation there is nothing to report back to the user. The arguments originally given are still in a single list and will be processed at the next stage as initially requested by the user.

In situations (i) and (ii) above, it will have been found that it is not possible to proceed in a manner as the user had originally wished. This would tend to suggest that the user was unfamiliar with the requirements to apply a test of the class requested. If the EXPLAIN facility is switched on, this is the default when the system is invoked, the requirements of the class of test involved are explained. The list of the requirements that have been applied are found in the class_checks array at the index position indicated by the testclass parameter. The procedure *expclassreqs* receives as a parameter a pointer to the first checknode in the chain, each of the checknodes are considered and a brief canned textual explanation appropriate to the semcheck keyword is then produced as shown below. The string passed to the parameter *testclassstr* is appropriate for the class of test involved and is either "measure of association" or "test of location".

```
WRITELN;
WRITELN('The requirement(s) for a ', testclassstr,
        'are as follows :-');
WRITELN;
WHILE ptocheck <> NIL
 DO WITH ptocheck ^ DO BEGIN
  CASE semcheck OF
   eqdomains :
    WRITELN('  Each sample should be measuring the ',
            'same quality or quantity.');
   relatedinst :
    WRITELN('  The instances of each sample should be related.');
   simenttype :
    WRITELN('  Each sample should be measured for ',
            'the same type of entity.')
   END;
   ptocheck := nextcheck
  END
```

At the end of the procedure reviewclasschecks, any listheadnodes and itemnodes referenced by the pointer invalidlists can be disposed of since they are of no further use. The validlists chain of listheadnodes are then assigned back to the listheadhead pointer. This pointer will be NIL if the system found that it would not be possible to apply a test to any group of arguments or if the user decided not to continue to try and apply a test with the arguments organised in the modified form. Otherwise the system will attempt to validate the use of a test for each of the groups of arguments represented by a listheadnode.

7.5 The Test Level Operation

The controlling procedure procstatreq inspects the pointer listheadhead and if it does not have the value NIL the procedure *checktestreq* is called, as below, to perform the test level operations.

```
CASE testclass OF
 association : checktestreq(testclass, assoc_checks, testname);
 location    : checktestreq(testclass, loc_checks, testname)
END;
```

The formal parameters of the procedure checktestreq are as follows :-

(i) testclass - this parameter has the value association or location to denote the class of test involved.

(ii) testchecks - an array of pointers referencing the semantic requirements of the tests of the class required, the array assoc_checks or loc_checks is passed as the actual parameter. The index of the testchecks array will be identical to that of the array being passed as the actual parameter. The conformant array parameters *firsttest* and *lasttest* can be used to identify the lower and upper bounds respectively of the testchecks array. These array bounds are used if it is required of the system to recommend a suitable test.

(iii) usertest - the value of this parameter is the name of the test that the user wishes to use, it has the value nulltest if no specific test was requested.

The objective of the procedure is to use the semantic requirements represented by the array testchecks to validate the use of a statistical test for each group of arguments. The overall operation of the procedure is given in the outline algorithm below.

```
WHILE another listheadnode to process
 DO BEGIN
   IF more than one group of arguments
    THEN display arguments currently being considered
   state := searching  { for a test }
   IF user has requested a specific test
    THEN BEGIN
     validate use of the test                                    --- A
     IF test is suitable
      THEN state := testfound
      ELSE BEGIN
       IF the explain facility is switched on
        THEN BEGIN
         list requirements of test                               --- B
         explain which requirement could not be met              --- C
        END
        ELSE just inform user that test cannot be applied
       does user wish to search for a test
       IF no THEN state := searchfailed
      END
    END
   IF state = searching
    THEN first test to consider has index firsttest
   WHILE state = searching
    DO BEGIN  { consider current test }
     validate use of the test                                    --- A
     IF test is suitable
      THEN BEGIN
       inform user of name of recommended test
```

```
          IF the explain facility is switched on
           THEN list requirements of test                                  --- B
          does user wish to apply recommended test
          IF yes
           THEN state := testfound
           ELSE state := searchfailed
         END
        ELSE IF test just considered had index lasttest
         THEN BEGIN
           state := searchfailed
           inform user that search has been unsuccessful
          END
        ELSE test to consider has the next index value in the array
     END
    IF state = testfound
     THEN review any data conversions etc                                 --- D
  END
```

Those parts of the algorithm identified by the labels A, B, C and D are explained in more detail in later sections of this chapter, a knowledge of their inner workings is not required at this stage of the test level description.

The procedure examines each listheadnode in turn and the arguments in the group about to be considered are displayed if there is more than one listheadnode in the chain. To record the current situation of the test level operation, regarding the arguments in the listheadnode under consideration, the variable *state* is used. If the system has yet to endorse the use of a test the current situation is *searching*, the initial value. Eventually either a test will be deemed appropriate, denoted by the value *testfound*, or the search for an acceptable applicable test will have failed, indicated by state having the value *searchfailed.*

If the user has requested a specific test to be used, i.e. the usertest parameter was not passed a value of nulltest, the system will first see if that test is appropriate. The procedure *validatetest* is called to check whether a particular test can be applied to a group of arguments (see section 7.6). If the data is suitable for the test, the state variable is updated to testfound. Otherwise the system informs the user that their preferred test is not suitable. If the EXPLAIN facility is set to on, the user is told of the requirements of the test (section 7.7) and given a reason as to why the one that failed could not be met (section 7.8), alternatively a simple message is output to the screen. The user will then be queried as to whether they wish the system to try and recommend an appropriate test. If the answer is no, the state variable is set to searchfailed (i.e. no test has been accepted for the data), otherwise state will still have the value searching.

The system will search for a test to recommend to the user either if the user's preferred test was unsuitable for the data or if no specific test was requested. As was mentioned earlier in section 6.3, the system will consider each test in the array testchecks according to their index order, the first index of the array is denoted by the conformant array parameter firsttest. The system will consider the tests in turn until either one is found to be acceptable or the end of the list is reached, identified by the index value lasttest. The current test under consideration is validated using the validatetest procedure.

If the test is found to be applicable the user is informed of the system's choice, a list of the requirements of the test is also given if the EXPLAIN facility is set to on. The system will then ask the user if they wish to accept the recommendation. If they do the process has been successful and the state variable is set to testfound, otherwise it is assigned the value searchfailed. This latter course of action is taken, to halt the search process once a recommended test has been rejected by the user, since it was decided that the system would only recommend one test. The alternative would be to continue through the list recommending other tests for which the requirements are met. The system will endorse the use of a user requested test that does not make the best use of the data, it may be that the user has a good reason for doing so. However, it was felt that if a user did not wish to accept a suggested test for some particular reason, and was not able to enter the name of an acceptable test, they would be best served by seeking the advice of a statistician to discuss the problem to be overcome. The existence of such problems that cannot be solved by the use of a predetermined strategy serves to highlight the fact that computer programs, no matter how good they are, will never replace human experts.

If the requirements of the test under consideration could not be met, the test with the subsequent index value would be the next candidate. If however the end of the list has been reached, i.e. the index value of the current test is the same as lasttest, the state variable is set to searchfailed and the user is informed that the data could not be manipulated into a form for a test to be applicable.

Having completed the above process for a group of arguments, successfully validating the use of a test may have involved transforming the data in some way to obtain a form suitable for the test requirements. If a suitable test has been found the system will report back to the user a summary of any transformations that have been applied to the data (section 7.9).

As is evident from the outline of the strategy adopted for the procedure checktestreq, if the system has to recommend to the user a test that is appropriate for the data it may be necessary to check the requirements of a number of tests before it may be possible to make a recommendation. To avoid the need to repeat the application of the same semantic check a number of times, the results of any checks that are applied to the current group of arguments are stored in a number of variables. When validating the use of a test the values of these variables can be examined, with it only being necessary to consult the information in the semantic data model to apply the check if the result is not already known. The variables are initialised in the checktestreq procedure each time a different listheadnode is considered, the procedures actually performing the semantic checks then update the variables as appropriate. Although primarily beneficial when validating the use of a test, these variables have also proved to be useful in the event of it being necessary to explain to the user why their requested test was inappropriate. The use of the variables is explained further in sections 7.6, 7.8 and 7.9.

7.6 Applying the Test Requirements

The procedure *validatetest* is called each time the controlling procedure checktestreq needs to validate the requirements of a particular test. The formal parameters of the procedure validatetest are as follows :-

(i) candtest - the test that the procedure is to validate;

(ii) ptocheck - a pointer to the head of the chain of checknodes identifying the requirements of the test being considered;

(iii) ptolisthead - a pointer to the listheadnode identifying the arguments that the test is to be validated for;

(iv) ptofailedcheck - the checknode of a requirement that cannot be met is referenced by this pointer, if all of the requirements are met and the end of the chain is reached the parameter will return with a value of NIL, signifying that the test is suitable.

As was stated in section 6.3, the checknodes will be considered in turn until either a requirement cannot be met or the end of the chain is reached. Whereas the objective at the class of test level was to group the arguments such that the requirements were met for each resultant group, at this level the system wishes to determine if the requirements are met for the arguments in the group identified by the current listheadnode.

As has already been noted, the program records the results of the semantic requirements performed on the current group being considered. The application of the semantic requirements and the manipulation of the data and the results variables is described in the following sections 7.6.1 to 7.6.6.

7.6.1 Related Samples

The procedure *checkrelargs* is called if the *argsrel* variable still has its initial value of *relunknown.* It determines as to whether the instances of the arguments in the group can be regarded as being related and will result in the argsrel variable being set to *related* or *unrelated* as appropriate. The requirement relatedinst is ratified if the argsrel variable has the value related.

7.6.2 Measurement

The measurement requirement of a test specifies the level of measurement demanded of the data and also any addition restriction that is placed upon the measurement schemes of the arguments involved. To validate such a requirement the level aspect will first be investigated before considering any constraint placed upon the measurement schemes used to record the data.

The variable *argsummary* is used to summarise the levels of measurement of the arguments in the ptolisthead group being considered, those levels distinguished are ratio, interval, rank, ordinal with a quantitative measurement scheme, ordinal with a qualitative measurement scheme, nominal with a closed set of categories and nominal with an open set of categories. The values that argsummary can take, to reflect the combination of levels, are as follows :-

(i) startstate - the initial value;

(ii) allrat - all ratio;

(iii) intrat - ratio or interval;

(iv) allqnt - ratio, interval or ordinal (quant);

(v) rankqnt - ratio, interval, rank or ordinal (quant);

(vi) rankqlt - ratio, interval, rank, ordinal (quant) or ordinal (qual);

(vii) ordqnt - ratio, interval, ordinal (quant) or ordinal (qual);

(viii) allord - all ordinal (qual);

(ix) nomqnt - ratio, interval, ordinal (quant), ordinal (qual) or nominal (closed);

(x) allqlt - ordinal (qual) or nominal (closed);

	ratio	interval	rank	ordinal (quant)	ordinal (qual)	nominal (closed)	nominal (open)
startstate	allrat	intrat	rankqnt	allqnt	allord	allqlt	novalidstate
allrat	allrat	intrat	rankqnt	allqnt	ordqnt	nomqnt	novalidstate
intrat	intrat	intrat	rankqnt	allqnt	ordqnt	nomqnt	novalidstate
allqnt	allqnt	allqnt	rankqnt	allqnt	ordqnt	nomqnt	novalidstate
rankqnt	rankqnt	rankqnt	rankqnt	rankqnt	rankqlt	novalidstate	novalidstate
rankqlt	rankqlt	rankqlt	rankqlt	rankqlt	rankqlt	novalidstate	novalidstate
ordqnt	ordqnt	ordqnt	rankqlt	ordqnt	ordqnt	nomqnt	novalidstate
allord	ordqnt	ordqnt	rankqlt	ordqnt	allord	allqlt	novalidstate
nomqnt	nomqnt	nomqnt	novalidstate	nomqnt	nomqnt	nomqnt	novalidstate
allqlt	nomqnt	nomqnt	novalidstate	nomqnt	allqlt	allqlt	novalidstate
novalidstate	novalidstate	novalidstate	novalidstate	novalidstate	novalidstate	novalidstate	novalidstate

Table 7.1 State Transition Table for Argsummary

(xi) novalidstate - the levels of measurement of the arguments are such that none of the tests can be applied, that is either there is a combination of rank and nominal level data or an argument has nominal (closed) data.

The initial value of startstate is assigned to argsummary in the checktestreq procedure when a new group is about to be processed. To compute the appropriate value of argsummary the attribute node of each argument is examined in turn to determine its level. The value of argsummary is then updated to reflect the data of those arguments considered thus far. The state transition table for the argsummary variable is shown in Table 7.1. The updated value of argsummary (given in the main body of the table) is dependent upon its current value (in the left hand margin) and the argument being considered (as shown in the top margin). For example, if the value of argsummary was currently allord and the next argument had a level of ordinal (quant), the value would be updated to ordqnt.

The value of the argsummary variable enables the system to determine if the level of measurement values are suitable for the test concerned. The reason for having a richer set of possible values than is necessary to accomplish this task is that the extra information can be used when considering any measurement scheme requirement.

The system may also have to examine the measurement schemes used to record the data before it can establish whether the measurement requirement can be met. The results of inspecting the schemes used and any attempts to convert the data into an appropriate form are recorded using the following variables :-

(i) qntdata - the result of examining the arguments for a test requiring quantitative data;

(ii) qltdata - the result of examining the arguments for a test requiring qualitative data;

(iii) dichdata - the result of examining the arguments for a test requiring dichotomous data.

Each variable will take one of the following values, indicating that :-

(i) dataunknown - the measurement schemes have not yet been examined for a level associated with the variable;

(ii) origOK - the original data is suitable for the requirement;

(iii) convOK - after performing some conversions the data is in an appropriate form, that is converted data has been generated for at least one of the arguments;

(iv) cannotconv - the data cannot be converted into an appropriate form for the level associated with the variable.

It was found to be advantageous to use three variables to record the results of examining the measurement schemes of the arguments as it provides a simple and complete means of recording what checks and conversions have been made.

To check whether the data of the arguments has been measured using, or can be converted to, the same measurement scheme the procedure *checksamemeas* is called. This procedure is used for the purpose of validating the requirements eqratqnt, eqintqnt, eqordqnt, eqordqlt, eqnomcat and eqdichcat, it has the following formal parameters :-

(i) ptolisthead - a pointer to the listheadnode under consideration.

(ii) measspec - the type of the measurement scheme that must be common to each argument, the value quant will be passed for the requirements eqratqnt, eqintqnt and eqordqnt; quant or ordqual for eqordqlt; ordqual or unordqual for eqnomcat; orddich or unorddich for eqdichcat. For the last three requirements the value passed will be dependent upon the argsummary value.

(iii) stateofdata - the result is returned via this parameter, the actual parameter will be qntdata, qltdata or dichdata which will initially have the value dataunknown.

If the arguments have not been measured with the same measurement scheme, one which is of an appropriate type, the user is prompted to enter the name of the measurement scheme to use, or NONE if it is not possible to convert all of the arguments to the same scheme. The user can enter the command SHOWARGMEAS to display the original measurement schemes used for the arguments and SHOWCANDMEAS to display those entries in the measurement directory that are of the type required. If the measurement scheme entered does not exist in the directory and the user wishes to declare it the appropriate procedure will be called to do so, otherwise the system will check that the chosen scheme is of the appropriate type. Each argument that has not been measured using the required scheme is then checked to see that the conversion is possible, which may necessitate the addition of entries to the conversion directory. If all of the conversions are possible the system would then generate the converted data as required. Upon leaving the procedure, the stateofdata parameter will have been set to origOK, convOK or cannotconv.

The system may also have to examine the measurement schemes of the arguments to validate the requirement nomcat, which requires that each argument be measured using a qualitative measurement scheme. If the argsummary value is allord or allqlt the requirement can be validated without any further work, if however some or all of the arguments are quantitative then the procedure *categoriseqnt* is called. The variable qltdata is passed as a parameter so that the result of trying to categorise the data can be returned. Each argument is examined and the user is prompted to enter the name of a qualitative measurement scheme for each of those that are quantitative, this may involve the addition of measurement and conversion schemes to the corresponding directories. If the data of an argument could not be converted into an appropriate form the result would be for the variable qltdata to return with the value cannotconv, otherwise it would have the value convOK with converted data having been generated where required.

The system is able to affirm the measurement requirement of a test using the argsummary variable and where relevant the variables qntdata, qltdata and dichdata. This is illustrated in the following section of code which is for the requirement eqnomcat.

```
eqnomcat :
 BEGIN
  IF (argsummary IN [allrat..allqnt, ordqnt..allord])
      AND (qltdata = dataunknown)
   THEN BEGIN
    checksamemeas(ptolisthead, ordqual, qltdata);
    IF qltdata = cannotconv THEN dichdata = cannotconv
   END
   ELSE IF (argsummary = allqlt) AND (qltdata = dataunknown)
    THEN BEGIN
     checksamemeas(ptolisthead, unordqual, qltdata);
     IF qltdata = cannotconv THEN dichdata := cannotconv
    END;
   testOK := qltdata IN [origOK, convOK]
 END;
```

7.6.3 Normality

The procedure *checknormalargs* checks the normality of the arguments in the group being considered, with the result being recorded using the variable *argsnormal*. For a number of statistical tests that assume normally distributed data, it is well known that under certain conditions sensible results can still be obtained with non-normal data, for example if the data is symmetrical. For this reason the user is allowed to insist on applying a test even if the system applied check for normality fails. The procedure works through the list of arguments and checks each for normality, if the check on an argument fails the user is informed and asked whether or not they wish to assume normality. In addition to giving a yes or no answer there is the alternative of accepting a default option, in which case the system errs on the side of caution and normality is not assumed. If an extra set of converted data has been generated for an argument the normality check is applied to that data, otherwise the original data referenced in the attribute node is used

If each argument passes the normal check the variable argsnormal will be assigned the value *normalOK*, if one or more arguments are assumed to be normal the value is *assnormal*, otherwise the value returned from the procedure checknormalargs will be *nonnormal*. The normality requirement is satisfied by either of the first two values.

7.6.4 Equality of Variances

As with the requirement that the data should be normally distributed, tests which assume that the variances of the samples involved are approximately equal can sometimes be applied when this is not the case, without unduly effecting the result. The user has the option of assuming that the variances of the samples are equal if the result of applying the relevant test of equal variances is significant, again the user can accept a system default (this does not assume equality). In calculating the variance of a sample a generated set of converted data is used instead of the original version.

The variable *argvar* records the result of checking the equality of the sample variances. The values that it can take once the check has been performed are *eqvarOK*, the check could find no significant difference, *asseqvar*, the user wishes to assume that the variances are equal, and *uneqvar*, in which case the requirement will not be satisfied.

7.6.5 Size of Samples

The variable *numinst* records the result of validating the requirement nige30. If the requirement is satisfied it will have the value *instOK*, otherwise the value assigned will be *insttoolow*.

7.6.6 Expected Frequencies in Contingency Tables

To validate the calculation of a chi-squared statistic on data to be arranged in a contingency table, it is necessary to examine the expected frequencies for each of the cells to ensure that they conform to the criteria given in section 6.4.9. The chi-squared statistic is used as a k-sample test of location and also forms the basis for two of the two-sample measures of association. The manner in which the contingency table would be formed is dependent upon whether it was being constructed for a measure of association or a test of location, as is illustrated in Figure 7.3.

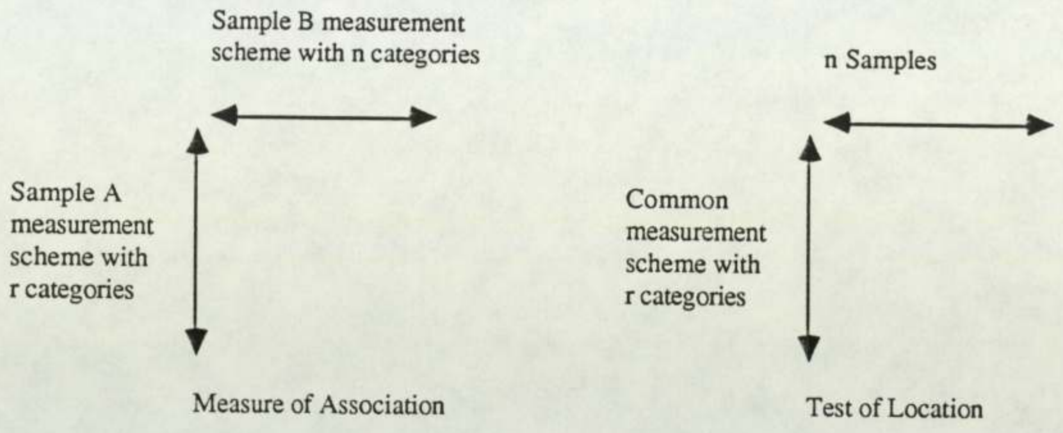
Sample B measurement scheme with n categories
Sample A measurement scheme with r categories
Measure of Association
n Samples
Common measurement scheme with r categories
Test of Location

Figure 7.3 Formation of a Contingency Table

The approach adopted by the system in performing the above mentioned validation is to compute the marginal totals of the table, from which the expected frequencies can be calculated. If the frequencies are too small the user is questioned as to whether any of the categories can be combined, in the case of a measure of association there may be two measurement schemes involved, for a test of location the samples are not combined. The process will continue until either the frequencies are acceptable or it is not possible to combine any more categories. Although the marginal totals have to be compiled differently depending upon the type of test involved, the process of validating the expected frequencies can be performed in a consistent manner.

To ratify the frequency requirement, the system computes and records the marginal frequency totals for the rows and columns of the virtual table, in addition to the information concerning which categories contribute to which total. The system is able to manipulate the categories, with the assistance of the user, to try and obtain marginal totals that produce acceptable expected frequencies. The contingency table information is represented in a form as shown in Figure 7.4, with the pointers *controw* and *contcolumn* referencing the row and column information respectively.

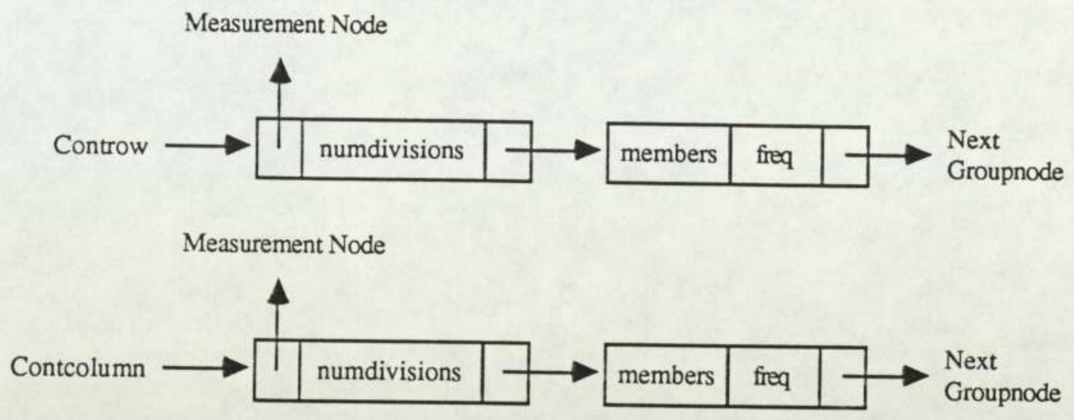

Figure 7.4 Representation of Contingency Table Marginal Frequencies

The controw and contcolumn pointers reference *groupinfo* nodes which in turn point to a chain of *groupnode* nodes. The fields of a groupinfo node are as follows :-

(i) measused - a pointer to the measurement scheme used to divide the dimension of the table, for the column of a test of location this pointer will be NIL since the marginal totals are formed for the number of items in each sample;

(ii) numdivisions - the number of marginal totals for the row or column;

(iii) grouphead - a pointer to the head of the chain of groupnodes.

There will be one groupnode for each marginal total of the dimension of the table being represented by the groupinfo node, with each node containing the following information :-

(i) members - if the dimension of the table has been divided according to measurement scheme categories, this field will identify the set of index numbers of the categories that contribute to the marginal total;

(ii) freq - an integer denoting the marginal total;

(iii) nextnode - a pointer to the next groupnode in the chain.

An integer variable, *conttotal*, is used to record the total frequency for the contingency table, i.e. the sum of the row and column marginal totals.

The expected frequencies can be calculated using the freq fields and the conttotal value. If the expected frequencies are too small the system searches for the set of categories with the smallest marginal total, provided that there are more than two divisions, and will ask the user if it can be combined with another, in the case of ordered categories this would be restricted to one that was adjacent. If the divisions can be combined the members and freq fields of the two nodes are added together and the redundant node is removed from the chain, the numdivisions field is also decremented. In the case of a measure of association where the same measurement scheme has been used for both arguments, it is necessary to adjust both the controw and contcolumn representations.

If combining the categories proved to be successful, the data would then be converted to reflect the grouping of the categories. If converted data had previously been generated for an argument, to satisfy a measurement requirement, it would be overwritten, otherwise an array would be generated to store it. In either case the measinfo pointer in the itemnode would not be changed.

The systems attempt at validating the chifreq requirement is recorded using the variable *argfreq*, this will take the value *freqOK* if successful and *freqtoolow* otherwise.

7.7 Explaining the Requirements of a Test

As at the class of test level, the system is able to produce portions of canned text to explain to the user the requirements for the use of a particular test. The procedure showtestreq is passed in the form of parameters the name of the test involved and also a pointer to the head of the chain of checknodes identifying the requirements of the test. The procedure works through the chain producing text appropriate for the semcheck keyword of each node. The procedure is used when the use of a user suggested test has not been validated or when the system is recommending a test to the user.

7.8 Explaining the Rejection of a User Requested Test

If the procedure validatetest determines that a particular test is unsuitable to be applied to a group of arguments, a pointer is returned which identifies the requirement that could not be met (the pointer has a value of NIL if the use of the test is endorsed). The procedure *expfailedcheck* is passed this pointer as a parameter and is able to explain to the user why their requested test could not be applied to the data. The semcheck keyword is used to produce a textual message relevant to the failed requirement. In the case of that being a measurement requirement, the system inspects the appropriate variable - qntdata, qltdata or dichdata - to determine whether the requirement was not met because the data could not be converted into an appropriate form or because the level of measurement values were unsatisfactory.

7.9 Reviewing the Validation of the Selected Test

Having selected a test for a group of arguments, the system reviews the results of any decisions made or actions performed during the process of validating the requirements of the test. The procedure *reviewtestchecks* works through the requirements of the selected test, it examines the associated variables recording the results of the procedures called to validate the requirements to decide if there is anything to report.

If the user has chosen to assume that a requirement is met, a message is output to warn that care should be taken when interpreting the results of the test. The assumptions that may have been made are that :-

(i) the data is normally distributed, i.e. argsnormal has a value of assnormal;

(ii) the variances of the samples are equal, signified by argvar having the value asseqvar.

If the EXPLAIN facility is set to on the system will also highlight any data conversions that have been made and why they were necessary. From the measurement requirement keyword and the corresponding result variable (qntdata, qltdata or dichdata), the system can deduce if the original measurement schemes of the arguments were unsuitable. A message is output to explain why the conversions were required, i.e. to obtain a common measurement scheme or to categorise quantitative data, in addition to a list of those arguments where converted data was generated (including the names of the measurement schemes of the original and converted data).

To satisfy the chifreq requirement some of the categories of the measurement scheme(s) involved may been combined. From the row and column representations of the contingency table information the system can list any groups of categories that have been formed.

Chapter 8

Conclusions

Having conducted the research, a number of conclusions can be drawn about the merit of using metadata in the development of knowledge based statistical software, further work that could be undertaken to more fully investigate the approach and some of its limitations. Some conclusions can also be made regarding the future for knowledge based statistical software in general.

8.1 The Prototype System

The objective of implementing a prototype system was to identify elements of metadata, in the form of semantic knowledge, such that a system could refer to the information to validate the use of a number of measures of association and tests of location. Having designed and implemented the prototype, it appears that by adopting such an approach it would be possible to enhance conventional style packages to provide a greater degree of statistical support and in doing so reduce the amount of misuse of statistical methods. Further research into the approach would therefore be worthwhile.

The prototype system allows the user to control the direction of the analysis, but monitors the commands issued to try and ensure that the data is appropriate and the results obtained will be meaningful. The checks that are performed, which see if the requirements for the correct use of a test are met, will result in one of the following :-

(i) the system cannot validate a requirement and informs the user that the test is not appropriate and cannot be applied, for example a request to apply a test requiring at least interval level data when some of the data is nominal;

(ii) the system cannot validate a requirement and asks the user if they wish to continue, for example in the case of the normality assumption;

(iii) the system validates the requirements and allows the test to be applied.

By including these alternatives the system has achieved the objective of preventing blatant misuse, yet giving a user the opportunity to make use of their knowledge of the situation to decide on subjective issues.

There remain a number of aspects of the prototype which could be further enhanced, a few areas of development are discussed below. Some of these features could not be incorporated into the prototype due to the time constraint placed upon the research whilst others are ideas that have come to light during the course of the implementation.

For the measures of association and tests of location that have been included, there may be other requirements that ought to be validated before their use is approved. The prototype system has predominantly been concerned with semantic issues, these may need to be more thorough, but the more numeric checks concerned with the actual data items have not been extensively covered. To take the paired samples t test as an example, Preece (1982) discussed the following topics which he thought should be considered before applying the test: outliers; homogeneity of the source of the data; trends; transformations; degree of precision of the data recording. It is fairly simple to add extra requirements, and the code to validate them, to the system. For some of the assumptions and requirements underlying the correct use of a test it may not be realistic for a system to attempt to check that they are met. It may be better to just remind the user of them, e.g. the independence of sample values, in the form of a checklist and having provided the information to leave it to the user to continue if they feel that none have been violated.

For those issues which are subjective, where the user should be given the option of continuing the process of validating the use of a test even though the system is unable to verify that a particular requirement is met, more information should be given to assist the user in making the decision. In the case of the normality assumption, textual information could be given to explain under which conditions normality is not too crucial, in addition to a plot of the data and the skewness and kurtosis coefficients.

The semantic knowledge about the data could also be used to enhance the explanation of the results obtained following the application of a statistical test or technique. Part of this could be to try and produce an explanation more oriented towards the ground domain of the user, the semantic knowledge encodes domain specific terminology and information to support this. If the user is able to understand the meaning underlying the result of the analysis, they will be able to determine if their initial hypothesis has been tested, it was noted in section 3.4 that the system does not attempt this task. Further research could extend the system to check that an analysis will be in accord with the aims of the user, e.g. does it matter that Pearson's PMCC can only detect a linear relationship.

Other enhancements that could be made to the prototype are more concerned with developing a 'proper' implementation, as would be required for a commercial system.

The data definition stage of declaring the required semantic knowledge about the data would need enhancing to provide the user with more assistance. For example, to use the prototype it is necessary to know what the level of measurement is for an attribute. The assistance that is given should be designed to be pedagogical.

The usual set of data manipulation functions should also be integrated. These would include editing and data selection functions, at present the data can only be specified using the *dsname.attname* format.

The data management aspect of statistical software was not considered at all in the prototype version. A commercial system would need to be able to handle large sets of data, missing data and possibly provide more data structures to facilitate the processing of more varied and complex data formats. The use of a database management system could be explored, one feature of this development would be to decide which part of the data model, if any, should be loaded into programming language data structures and what information should be read from the database each time it was needed.

To produce a system with a practical use it would of course be necessary to extend the areas of statistics covered. It would have to be further substantiated that statistical software based on a semantic modelling approach could be developed to provide support to the range of statistical facilities required of a general purpose statistical package.

8.2 Assessment of Using Metadata

The majority of the research that has been undertaken in the area of knowledge based statistical software has been to develop systems based upon a consultation with a professional statistician. In order to ascertain whether or not a recommendation can be made, such systems engage in a discourse with the user. The user is questioned for information as and when it is required, building up a picture of the data a fragment at a time. By way of contrast, the research has considered an approach whereby a model of the data is declared, which contains the knowledge of the data that may be required.

An advantage of having a single data definition stage to make known the metadata is that it allows the data model to be declared for the users by local experts, those with a knowledge of both the domain of application and statistics. When using the system to perform a statistical analysis, a statistically naive user would be spared any questions regarding the statistical nature of the data, for example the level of measurement. The system would be able to validate any data that the users may have enter and also verify that requests to apply statistical techniques are appropriate. A separate data definition stage would also seem particularly suited to situations where repeated trials or surveys are involved. The metadata content will typically remain unaltered, to a great extent, and can act as a template for the structure of any new data. In the case of large sets of data, the metadata model would also serve as a good source of documentation.

A further advantage of defining the metadata beforehand is that the system has access to some domain specific terminology and knowledge. The need to combine statistical and ground domain knowledge was identified as being one of the major problems in developing knowledge based statistical software.

8.3 Extensions to the Metadata

The semantic knowledge that has been represented in the data model of the prototype seems to have a general utility and would be commonly used to validate the requirements of a range of statistical methods. Further work could investigate in what way the metadata content of the model could be usefully extended.

A number of possible areas of metadata that could be incorporated are :-

(i) At the data level.

The levels of measurement identified could be extended to include counts as a separate category, they can be regarded as being more specialised than ratio level data.

(ii) At the attribute level.

Functional relationships that can be identified between attributes could be represented. One attribute could be related to, or computed from, one or more other attributes.

(iii) At the dataset level.

It may prove to be profitable to include metadata about the source of the data or any sampling procedures that have been used. For example, the entity instances of a particular entity type can be regarded as a set, with the specific entities of a dataset being a subset. A subset may have been selected because of the value or values of one or more of the properties of the entities, e.g. a sample of salesmen aged over 40.

The investigation to determine what extra metadata ought to be added to the data model could be aided by focussing on a number of application areas. This could result in :-

(i) the identification of further items of generally useful metadata;

(ii) the recognition that different application areas have different metadata needs and models tailored to the application would be more suitable.

The initial aim was to work towards the development of a general purpose data model, however this will not be possible if the metadata has to be biased because of the nature of the data involved or due to the favoured statistical techniques that are employed.

As has been argued earlier in the thesis, it would not be practical to continue adding further to the data model ad infinitum, the problem is deciding what extra knowledge should be added and where to draw the line.

8.4 Limitations of a System Using Metadata

As the metadata content of a data model is extended, the likelihood of requiring the declaration of information that is not subsequently used also increases. This problem does not occur if the information is acquired by the system incrementally. A solution would be to have a core section of the metadata that must be given prior to entering the actual data, leaving it optional to declare the remainder. If the system required a piece of information that was marked as unknown, the user would be asked to supply the relevant metadata such that it could be added to the model.

A problem would occur if the time taken to search the metadata and validate the requirements had adverse effects on the response time of the system. With the prototype there is no noticeable degradation, indicating that there is room for further processing to be carried out.

The motivation for adopting an approach that could enhance conventional style command driven packages was to provide a flexible tool to do general data analysis. A system performing such a function would seem to be able to use metadata to try and ensure that the statistical methods are not misused. For areas such as experimental design or the application of advanced statistical techniques (e.g. regression or time series analysis), a metadata approach may have to be supplemented with other knowledge.

8.5 The Future for Knowledge Based Statistical Software

Much of the research that has been carried out in the area of knowledge based statistical software has concentrated on the development of what the artificial intelligence community have termed expert systems. These systems have generally sought to encode a strategy to guide a user to perform a particular task by taking the role of a consultant statistician, such software seems particularly suited to these tasks. Although as yet none have become commercially available, the prevailing feeling at the Compstat 88 conference (which is one of the most prestigious conferences devoted to computational statistics) was that some systems would appear on the market within 12 months.

Although useful expert systems will become available in the near future, work should continue to develop other forms of knowledge based statistical software. Expert systems should not be seen as the solution, but as one of a number of methods by which software is able to provide statistical support and guidance. Research should continue to develop systems based on other architectures, to provide other forms of assistance by adopting other roles.

At Compstat 88, it was noticeable that there was a tendency to avoid the expression expert systems and to use instead the term consultancy systems. Nelder (1988) reported a realisation that systems would not be capable of being authoritarian and should instead be libertarian. Although software will almost certainly never achieve the level of expertise of a human statistician, it is possible to improve significantly on the packages that are currently being used, many of which have their roots in the 1960's.

References

Altman, D.G. (1982), "Statistics in Medical Journals", *Statistics in Medicine* **1**, pp. 59-71.

Anscombe, F.J. (1960), "Examination of Residuals", *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 1-36.

Badgley, R.F. (1961), "An assessment of Research Methods Reported in 103 Scientific Articles from Two Canadian Medical Journals", *Canadian Medical Association Journal* **85**, pp. 246-250.

Baines, A. & Clithero, D.T. (1986), "Interactive User-Friendly Package for Design and Analysis of Experiments", *COMPSTAT 86* (7th Symposium, Rome, Italy), Physica-Verlag, pp. 320-325.

Bartlett, M.S. (1937), "Properties of Sufficiency and Statistical Tests", *Proceedings of the Royal Society Series A* **160**, pp. 268-282.

Becker, R.A. & Chambers, J.M. (1984), *S: An Interactive Environment for Data Analysis and Graphics*, Wadsworth Advanced Book Program.

Blum, R.L. (1982), *Discovery and Representation of Causal Relationships from a Large Time-Orientated Clinical Database: The RX Project*, Springer-Verlag.

Box, G.E.P. (1953), "Non-Normality and Tests on Variances", *Biometrika* **40**, pp. 318-335.

Brachman, R.J. (1983), "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks", *Computer* **16**(10), pp. 30-36.

Chambers, J.M. (1980), "Statistical Computing: History and Trends", *The American Statistician* **34**(4), pp. 238-243.

Chambers, J.M. (1981), "Some thoughts on Expert Software", *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, Springer-Verlag, pp. 36-40.

Chen, P.P.S. (1976), "The Entity-Relationship Model - Toward a Unified View of Data", *ACM Transactions on Database Systems* **1**(1), pp. 9-36.

Codd, E.F. (1979), "Extending the Database Relational Model to Capture More Meaning", *ACM Transactions on Database Systems* **4**(4), pp.397-434.

Conover, W.J. (1973), "Rank Tests for One Sample, Two Samples, and K Samples Without the Assumption of a Continuous Distribution Function", *The Annals of Statistics* **1**(6), pp. 1105-1125.

Conover, W.J. (1980), *Practical Nonparametric Statistics* (2nd Edition), Wiley.

Date, C.J. (1983), "The Extended Relational Model RM/T", in *An Introduction to Database Systems: Volume II*, Addison-Wesley, pp. 241-289.

Date, C.J. (1986), "Semantic Modelling", in *An Introduction to Database Systems: Volume I* (4th Edition), Addison-Wesley, pp.609-623.

Dickson, J.M. (1984), "Data Capture and Validation Using Portable Terminals", *COMPSTAT 84* (6th Symposium, Prague, Czechoslovakia), Physica-Verlag, pp. 473-478.

Dickson, J.M. & Talbot, M. (1986), "Statistical Data Validation and Expert Systems", *COMPSTAT 86* (7th Symposium, Rome, Italy), Physica-Verlag, pp. 283-288.

Gale, W.A. (1985), "Knowledge Representation in Data Analysis", *Proceedings of the Fourth International Symposium on Data Analysis and Informatics* (Versailles, France), North-Holland, pp. 703-719.

Gale, W.A. (1986a), "Overview of Artificial Intelligence and Statistics", in Gale, W.A. (ed), *Artificial Intelligence and Statistics*, Addison-Wesley, pp. 1-16.

Gale, W.A. (1986b), "Student Phase 1 - A Report on Work in Progress", in Gale, W.A. (ed), *Artificial Intelligence and Statistics*, Addison-Wesley, pp. 239-265.

Gale, W.A. & Pregibon, D. (1982), "An Expert System for Regression Analysis", *Computer Science and Statistics: Proceedings of the 14th Symposium on the Interface* (New York, USA, July 1982), Springer-Verlag, pp. 110-117.

Gale, W.A. & Pregibon, D. (1984), "Constructing an Expert System for Data Analysis by Working Examples", *COMPSTAT 84* (6th Symposium, Prague, Czechoslovakia), Physica-Verlag, pp. 227-236.

Glantz, S.A. (1961), "Biostatistics: How to Detect, Correct and Prevent Errors in the Medical Literature", *Circulation* **61**(1), pp. 1-7.

Gore, S.M., Jones, I.G. & Rytter, E.C. (1977), "Misuse of Statistical Methods: Critical Assessment of Articles in BMJ from January to March 1976", *British Medical Journal* **1**, pp. 85-87.

Hahn, G.J. (1985), "More Intelligent Statistical Software and Statistical Expert Systems: Future Directions", *The American Statistician* **39**(1), pp. 1-16.

Hajek, P. & Havranek, T. (1978), "The GUHA Method - Its Aims and Techniques", *International Journal of Man-Machine Studies* **10**, pp. 3-22.

Hajek, P. & Ivanek, J. (1982), "Artificial Intelligence and Data Analysis", *COMPSTAT 82* (5th Symposium, Toulouse, France), Physica-Verlag, pp. 54-60.

Hammond, R.G. (1983), "RAPID: A Statistical Database Management System", *Computer Science and Statistics: Proceedings of the 15th Symposium on the Interface* (Texas, USA, March 1983), North-Holland, pp. 31-34.

Hand, D.J. (1984), "Statistical Expert Systems: Design", *The Statistician* **33**, pp.351-369.

Hand, D.J. (1985a), "Statistical Expert Systems: Necessary Attributes", *Journal of Applied Statistics* **12**(1), pp.19-27.

Hand, D.J. (1985b), "Choice of Statistical Technique", Bulletin of the International Statistical Institute (Proceedings of the 45th Session, Vol. 3, Amsterdam, August 1985), pp. 21.1-1 to 21.1-16.

Hand, D.J. (1986), "Expert Systems in Statistics", *The Knowledge Engineering Review* **1**(3), pp. 2-10.

Hand, D.J. (1987), "A Statistical Knowledge Enhancement System", *Journal of the Royal Statistical Society Series A* **150**(4), pp. 334-345.

Haux, R. & Jöckel, K.H. (1986), "Database Management and Statistical Data Analysis: The Need for Integration and for Becoming More Intelligent", *COMPSTAT 86* (7th Symposium, Rome, Italy), Physica-Verlag, pp. 407-414.

Hooke, R. (1980), "Getting People to Use Statistics Properly", *The American Statistician* **34**(1), pp. 39-42.

Huber, P.J. (1986), "Environments for Supporting Statistical Strategy", in Gale, W.A. (ed), *Artificial Intelligence and Statistics*, Addison-Wesley, pp. 285-294.

Jones, B. (1980), "The Computer as a Statistical Consultant", *Bulletin in Applied Statistics* **7**(2), pp. 168-195.

Kendall, M. & Stuart, A. (1979), *The Advanced Theory of Statistics Volume 2: Inference and Relationship* (4th Edition), Charles Griffin & Co.

Lehmann, E.L. (1975), *Nonparametrics: Statistical Methods Based On Ranks*, Holden-Day.

Lundy, R.T. (1984), "Metadata Management", *Database Engineering*, **7**(1), pp. 43-48.

Marascuilo, L.A. & McSweeney, M. (1977), *Nonparametric and Distribution-Free Methods for the Social Sciences*, Brooks/Cole.

McCarthy, J.L. (1982), "Metadata Management for Large Statistical Databases", *Proceedings of 8th International Conference on Very Large Databases* (Mexico), pp. 234-243.

Nelder, J.A. (1977), "Intelligent Programs, The Next Stage in Statistical Computing", in Barra, J.R. (ed), *Recent Developments in Statistics*, North-Holland, pp.79-86.

Nelder, J.A. (1984), "Present Position and Potential Developments: Some Personal Views, Statistical Computing", *Journal of the Royal Statistical Society Series A* **147**(2), pp. 151-160.

Nelder, J.A. (1988), "How Should the Statistical Expert System and its User See Each Other ?", *COMPSTAT 88* (8th Symposium, Copenhagen, Denmark), Physica-Verlag, pp. 107-116.

Oldford, R.W. & Peters, S.C. (1984), "Building a Statistical Knowledge Based System with Mini-Mycin", *Proceedings of the ASA: Statistical Computing Section*, pp. 85-90.

Oldford, R.W. & Peters, S.C. (1986), "Implementation and Study of Statistical Strategy", in Gale, W.A. (ed), *Artificial Intelligence and Statistics*, Addison-Wesley, pp. 335-353.

Portier, K.M. & Lai, P. (1983), "A Statistical Expert System for Analysis Determination", *Proceedings of the ASA: Statistical Computing Section*, pp. 309-311.

Preece, D.A. (1982), "t is for Trouble (and Textbooks): A Critique of Some Examples of the Paired-Samples t-test", *The Statistician* **31**(2), pp.169-195.

Pregibon, D. & Gale, W.A. (1984), "REX: An Expert System for Regression Analysis", *COMPSTAT 84* (6th Symposium, Prague, Czechoslovakia), Physica-Verlag, pp. 242-248.

Quillian, M.R. (1966), "Semantic Memory", Report AFCRL-66-189, Bolt Beranek and Newman, Cambridge, Massachusetts, USA.

Schor, S. & Karten, I. (1966), "Statistical Evaluation of Medical Journal Manuscripts", *Journal of the American Medical Association* **195**(13), pp. 1123-1128.

Shapiro, S.S. & Wilk, M.B. (1965), "An Analysis of Variance for Normality (Complete Samples)", *Biometrika* **52**, pp. 591-611.

Siegel, S. (1956), *Nonparametric Statistics for the Behavioral Sciences*, McGraw-Hill.

Shortliffe, E.H. (1976), *Computer-based Medical Consultations: MYCIN*, Elsevier Scientific.

Smith, A.M.R., Lee, L.S. & Hand, D.J. (1983), "Interactive User-friendly Interfaces to Statistical Packages", *The Computer Journal* **26**(3), pp. 199-204.

Stevens, S.S. (1946), "On the Theory of Scales of Measurement", *Science*, **103**(2684), pp. 677-680.

Wolstenholme, D.E. & Nelder, J.A. (1986), "A Front End for GLIM", in Haux, R. (ed), *Expert Systems in Statistics*, Gustav Fischer, pp. 155-177.

Appendix A
Example

The text of the thesis describes the knowledge that is represented in the data model, the requirements that can be specified for a type of test or a particular test to be valid and how the system uses the semantic knowledge and the data to validate the various requirements. The system has been tested to ensure that it runs as specified and this example is given merely for completeness.

Figure A.1 illustrates the entity taxonomy that has been declared in the example data model, showing the organisation of the entity types and the relationships between them. The three data sets that have been declared (sec_info, clerk_info and eng_info) are also displayed together with their respective attributes, more of the content of the example data model is given in the trace of the program running.

Finally, the trace gives the outcome of a request to perform a statistical test.

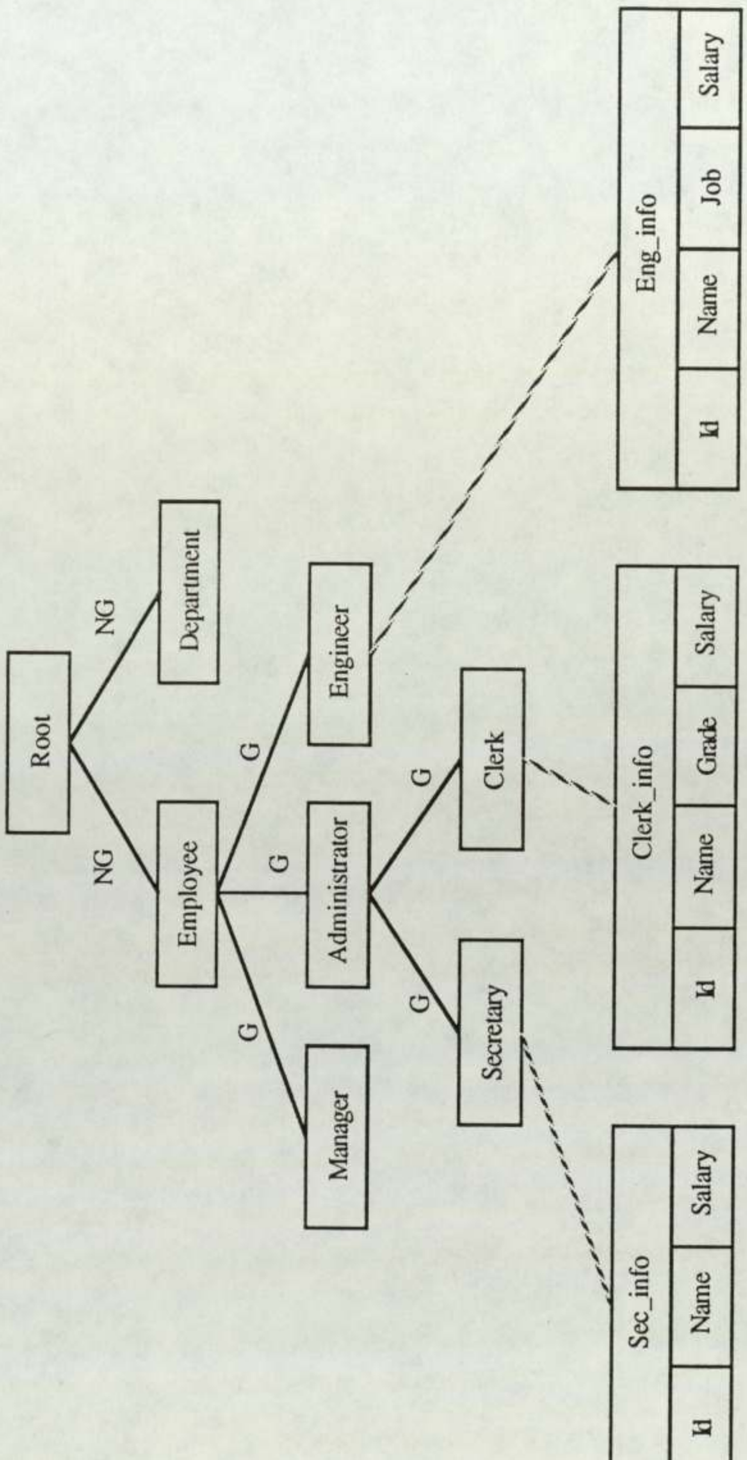

Figure A.1 Entity Taxonomy and Datasets Used in Example

```
$ runmain

> showentdir root

Entity type                  Relation to super type

ROOT                                   NOT_APPLIC
  DEPARTMENT                           NONGENERIC
  EMPLOYEE                             NONGENERIC
    ENGINEER                              GENERIC
    ADMINISTRATOR                         GENERIC
      CLERK                               GENERIC
      SECRETARY                           GENERIC
    MANAGER                               GENERIC

> showdsdir

Data set          Entity type       Instances

CLERK_INFO        CLERK                12
ENG_INFO          ENGINEER             10
SEC_INFO          SECRETARY            15
```

```
> showatt clerk_info

Att name          Att role  Att type            Level          Meas

ID                     KEY  EMP_ID            NOMINAL          NAT_INS_NO
NAME               NON_KEY  EMP_NAME          NOMINAL          NAME
GRADE              NON_KEY  GRADE             ORDINAL          CLERIC_GRADE
SALARY             NON_KEY  SALARY              RATIO          POUNDS_PER_YEAR

> showatt eng_info

Att name          Att role  Att type            Level          Meas

ID                     KEY  EMP_ID            NOMINAL          NAT_INS_NO
NAME               NON_KEY  EMP_NAME          NOMINAL          NAME
JOB                NON_KEY  JOB_TYPE          NOMINAL          ENG_TYPE
SALARY             NON_KEY  SALARY              RATIO          POUNDS_PER_YEAR

> showatt sec_info

Att name          Att role  Att type            Level          Meas

ID                     KEY  EMP_ID            NOMINAL          NAT_INS_NO
NAME               NON_KEY  EMP_NAME          NOMINAL          NAME
SALARY             NON_KEY  SALARY              RATIO          POUNDS_PER_MONTH
```

```
> showmeasdir

Meas name          Meas type

CLERIC_GRADE        QUALMEAS
ENG_TYPE            QUALMEAS
NAME                QUALMEAS
NAT_INS_NO          QUALMEAS
POUNDS_PER_MONTH   QUANTMEAS
POUNDS_PER_YEAR    QUANTMEAS

> showmeas nat_ins_no

  Meas type = qualitative,  Set type = open,  Data type = IDENTIFIER

> showmeas cleric_grade

  Meas type = qualitative,  Categories are ordered

  Valid category values are :-
    CLERIC_1
    CLERIC_2
    CLERIC_3
    CLERIC_4
```

```
> showmeas eng_type

  Meas type = qualitative,  Categories are unordered

  Valid category values are :-
    MECHANICAL
    ELECTRICAL

> showmeas pounds_per_year

  Meas type = quantitative
  Lower bound =        0.00
  Upper bound = max

> randomised_block sec_info.salary eng_info.salary :
+ clerk_info.salary

The requirements for RANDOMISED_BLOCK are as follows :-

  The instances of each sample should be related
  The data should be measured on at least an
    interval scale using the same measurement scheme
  The data in each sample should be normally distributed
  The variance of each sample should be equal

The requested test was not acceptable because the arguments
  are not related
```

```
Do you wish to search for a test of location (yes/no) : yes

Each argument needs to be measured with the same
  quantitative measurement scheme
Enter measurement scheme to use (or NONE) : showargmeas
SEC_INFO.SALARY                     POUNDS_PER_MONTH
ENG_INFO.SALARY                     POUNDS_PER_YEAR
CLERK_INFO.SALARY                   POUNDS_PER_YEAR
Enter measurement scheme to use (or NONE) : pounds_per_year
Is it possible to convert from POUNDS_PER_MONTH
                           to   POUNDS_PER_YEAR
Answer yes/no : yes
Enter multiplying factor : 12
Enter constant : 0

Do you wish to assume that the data in ENG_INFO.SALARY
  is normally distributed (yes/no/default) : yes

Recommended test of location is      ONE_WAY_AOV

The requirements for      ONE_WAY_AOV are as follows :-

  The data should be measured on at least an
    interval scale using the same measurement scheme
  The data in each sample should be normally distributed
  The variance of each sample should be equal
```

```
Do you wish to apply this test (yes/no) : yes

The data has been converted to the same measurement scheme
SEC_INFO.SALARY converted from POUNDS_PER_MONTH to POUNDS_PER_YEAR

Warning, care should be taken when interpreting
  the results since the test assumes that the
  data is normally distributed

Call proc to apply       ONE_WAY_AOV

> quit
Exiting from program
```

Appendix B
Program Listings

Given on the following two pages is the organisation of the procedures which constitute the program code, grouped according to the relevant section of the system. The remainder of the appendix lists the files which comprise the total implementation, the files and contents being :-

(i) main.pas - the majority of the type declarations, the auxiliary routines and the main program;

(ii) model_routines.pas - the model management system;

(iii) check_routines.pas - the statistics validation system;

(iv) keyworddir.dat - a list of the character strings and corresponding enumeration type values for the reserved words of the command language;

(v) classcheckdir.dat - the classes of tests requirements;

(vi) assoccheckdir.dat - the measures of association requirements;

(vii) loccheckdir.dat - the tests of location requirements;

(viii) shapwilkcoeff.dat - the coefficients for the Shapiro-Wilk test of normality.

AUXILIARY		skipwhitespace, skipblankchar, gettoken, reporterror, setup
MODEL MANAGEMENT	Entity Taxonomy	etypesearch, addetype, etypeaddition, displayentdir
	Dataset Directory	dstypesearch, adddstype, dstypeaddition, displaydsdir
	Measurement Directory	meassearch, addmeasscheme, catsearch, newcatnode, getqualclasses, getqualinfo, getquantinfo, measaddition, meastypeOK, displaymeas, displaymeasdir, displaycandmeas, wantstodecmeas
	Conversion Directory	convsearch, addconvscheme, genqntqltnode, genqltqltnode, getconvinfo, performconv
	Attributes	atttypesearch, addattnode, atttypeaddition, displayatt
	Instances	instaddition, charcatvalue, numcatvalue, displayinst
	Backing Store	loadkeywords, loadsemchecks, loadmeasdir, loadconvdir, loadenttree, loadattlist, loaddstree, loadknowbase, saveetype, savedstype, saveattlist, savedata, savemeasscheme, saveconvscheme

STATISTICS VALIDATION	Checks	checkeqdom, checkenttype, checkrelinsts, checkrelargs, checknumargs, checknormalargs, Ftest, Bartlett, Box, checkeqvar, checknige30, addcontnode, setupcontnodes, setcontfreqs, formassoccont, formloccont, efreqOK, findminfreq, combgroups, trytocombgroups, regroupdata, combassoccat, combloccat, checkchifreq, performsummary
	Data Conversion	convertdata, checksamemeas, disposegenqlt, catogoriseqnt, dichqltdata
	Test Level	validatetest, showtestreq, expfailedcheck, displaycombcats, displayconvargs, reviewtestchecks, disposecontinfo, checktestreq
	Type of Test Level	removeitem, createlist, addtolist, disposeoflist, checkclassreq, expclassreqs, reviewclasschecks
	Preliminary and Control	genitemnode, procstatreq

B.1 Main.pas

```
PROGRAM MAIN (INPUT, OUTPUT, keyworddir, checkdir, entdir,
              dsdir, attdir, datafile, measdir, convdir,
              shapwilkcoeff);


CONST wordlength = 16;
      doublelength = 32;
      nullname = '                ';
      messagelength = 30;
      keywordmax = 100;
      datalength = 50;
      prompt = '> ';
      contprompt = '+ ';
      contchar = ':';
      underscore = '_';
      minreal = -1E7;
      maxreal = 1E7;

TYPE  word = PACKED ARRAY [1..wordlength] OF CHAR;
      doubleword = PACKED ARRAY [1..doublelength] OF CHAR;
      numarray = ARRAY [1..datalength] OF REAL;
      chararray = ARRAY [1..datalength] OF word;
      textmessage = VARYING [messagelength] OF CHAR;
      errortype = (continue, attdup, attexist, attmiss, dsexists,
        dsmiss, entexists, entmiss, eqexp, identexp, insuffinst,
        invarg, invinfo, invlevel, invmeas, invmeastyp,
        invnumarg, invrel, invval, measexists, measmiss, noatts,
        noinsts, nolevel, nomeas, notype, numexp);
      validtokens = (addatt, adddds, addent, addinst, addmeas,
        showargmeas, showatt, showcandmeas, showdsdir, showentdir,
        showinst, showmeas, showmeasdir, exptok, noexptok, quit,
        association, location,
        pearson, spearman, kendall, tau_c, cramers_v, coeff_of_cont,
        normal_test, t_paired, randomised_block, t_common,
        t_separate, one_way_aov, wilcoxon, sign_test, friedman_aov,
        mann_whitney, kruskal_wallis, mcnemar_test, cochran_q,
        chi_squared, fisher_exact, nulltest,
        nongentok, gentok, opentok, closedtok,
        yestok, notok, deftok,
        typetok, leveltok, meastok, normtok,
        qualmeas, quantmeas, chartok, numtok,
        nomtok, ordtok, ranktok, inttok, rattok,
        endofline, assign, dot, identifier, numeral, endofinfo,
        min, max, upper, nonetok, errtoken);
      statcomms = association..fisher_exact;
      valid_tests = pearson..nulltest;
      classtype = association..location;
      assoctype = pearson..coeff_of_cont;
      loctype = normal_test..fisher_exact;
      validreqs = (twosample, ksample, eqdomains, relatedinst,
        simenttype, normal, eqvar, nige30, chifreq,
        eqratqnt, ratqnt, eqintqnt, intqnt, ranked, eqordqnt,
        eqordqlt, ordqlt, eqnomcat, nomcat, eqdichcat );
      testtype = twosample..ksample;
      taxon_relation = (not_applic, nongeneric, generic);
      role_type = (key, non_key);
      data_levels = (none, rank, nominal, ordinal, interval, ratio);
      meas_level = qualmeas..quantmeas;
      qualsettype = (setunknown, openset, closedset);
```

```
qualordtype = (ordunknown, unordered, ordered);
datatype = identifier..numeral;
sortofmeas = (quant, ordqual, unordqual, qual, orddich,
  unorddich, dich);
convtype = (qnt_qnt, qnt_qlt, qlt_qlt);
enodepointer = ^ e_node;
dsnodepointer = ^ ds_node;
attnodepointer = ^ att_node;
numpointer = ^ numarray;
charpointer = ^ chararray;
measpointer = ^ meas_node;
catnodepointer = ^ cat_node;
qntqntpointer = ^ qntqntnode;
qntqltpointer = ^ qntqltnode;
qltqltpointer = ^ qltqltnode;
checkpointer = ^ checknode;
convpointer = ^ conv_node;
keywordentry = RECORD
  keystr   : word;
  keytoken : validtokens
END;
e_node = RECORD
  ent_name    : word;
  super_rel   : taxon_relation;
  superpointer,
  subpointer,
  nextpointer : enodepointer
END;
ds_node = RECORD
  ds_name   : word;
  leftp,
  rightp    : dsnodepointer;
  ent_type  : enodepointer;
  instances : INTEGER;
  attchain  : attnodepointer
END;
att_node = RECORD
  att_name  : word;
  next_att  : attnodepointer;
  att_role  : role_type;
  att_type  : word;
  datalevel : data_levels;
  att_dist  : (normaldist, distunknown);
  meas_p    : measpointer;
  CASE mode : datatype OF
    identifier : (char_p : charpointer);
    numeral    : (num_p  : numpointer)
END;
meas_node = RECORD
  measname : word;
  leftp,
  rightp   : measpointer;
  CASE meas_type : meas_level OF
    qualmeas :
      (cattype  : datatype;
       settype  : qualsettype;
       ordtype  : qualordtype;
       numofcat : INTEGER;
       cathead  : catnodepointer);
    quantmeas :
      (lowerbound,
       upperbound : REAL)
END;
```

```
      cat_node = RECORD
        next : catnodepointer;
        CASE cattype : datatype OF
          identifier : (charvalue : word);
          numeral    : (numvalue  : REAL)
      END;
      conv_node = RECORD
        from_to : doubleword;
        left_p,
        right_p : convpointer;
        CASE typeofconv : convtype OF
          qnt_qnt : (qntqnt_p : qntqntpointer);
          qnt_qlt : (qntqlt_p : qntqltpointer);
          qlt_qlt : (qltqlt_p : qltqltpointer)
      END;
      qntqntnode = RECORD
        a,
        c : REAL
      END;
      qntqltnode = RECORD
        upper : REAL;
        index : INTEGER;
        next  : qntqltpointer
      END;
      qltqltnode = RECORD
        toindex : INTEGER;
        next    : qltqltpointer
      END;
      checknode = RECORD
        semcheck  : validreqs;
        nextcheck : checkpointer
      END;
      tokeninfo = RECORD
        CASE ttype : validtokens OF
         identifier : (tchars : word);
         numeral    : (tnum : REAL);
         OTHERWISE    ()
      END;

VAR   keywordtable : ARRAY [1..keywordmax] OF keywordentry;
      numofkeywords : INTEGER;
      ent_root : enodepointer;
      ds_root : dsnodepointer;
      meas_root : measpointer;
      conv_root : convpointer;
      class_checks : ARRAY [classtype] OF checkpointer;
      assoc_checks : ARRAY [assoctype] OF checkpointer;
      loc_checks : ARRAY [loctype] OF checkpointer;
      lowercase, uppercase, digits, letters,
      wordchars, numberstart : SET OF CHAR;
      token : tokeninfo;
      explain : BOOLEAN;


PROCEDURE skipwhitespace;

BEGIN
 WHILE (INPUT ^ = ' ') OR (INPUT^ = contchar)
  DO BEGIN
   IF EOLN(INPUT) THEN WRITE(prompt);
   GET(INPUT)
  END
END; { proc skipwhitespace }
```

```
PROCEDURE skipblankchar;

BEGIN
 WHILE ((INPUT ^ = ' ') OR (INPUT ^ = contchar))
       AND (NOT EOLN(INPUT))
  DO IF INPUT ^ = contchar
   THEN BEGIN
    WRITE(contprompt);
    READLN
   END
   ELSE GET(INPUT)
END; { proc skipblankchar }


FUNCTION strlen (
  VAR string : PACKED ARRAY [lower..upper : INTEGER] OF CHAR ) :
INTEGER;

VAR index : INTEGER;

BEGIN
 strlen := 0;
 FOR index := lower TO upper
  DO IF string[index] <> ' ' THEN strlen := index
END; { funct strlen }


PROCEDURE gettoken;

  PROCEDURE checkkeywords;

  VAR index : INTEGER;

  BEGIN
   index := 1;
   WHILE (token.ttype = identifier) AND (index <= numofkeywords)
    DO WITH keywordtable[index] DO BEGIN
     IF token.tchars = keystr
      THEN token.ttype := keytoken;
     index := index + 1
    END;
   IF token.ttype = min
    THEN BEGIN
     token.ttype := numeral;
     token.tnum := minreal
    END
   ELSE IF token.ttype = max
    THEN BEGIN
     token.ttype := numeral;
     token.tnum := maxreal
    END
  END; { proc checkkeywords }

  PROCEDURE readword;

  { reads a string of text into token.tchars,
    valid characters are letters, digits and underscore with
    letters being converted to uppercase, if the string
    is longer than wordlength the remaining characters
    are passed over and ignored }
```

```
VAR ch : CHAR;
    i,
    strlength : INTEGER;

BEGIN
 token.ttype := identifier;
 token.tchars := nullname;
 strlength := 0;
 ch := INPUT ^;
 WHILE (ch IN wordchars) AND (strlength < wordlength)
  DO BEGIN
   strlength := strlength + 1;
   IF ch IN lowercase
    THEN token.tchars[strlength] := CHR(ORD(ch) - 32)
    ELSE token.tchars[strlength] := ch;
   GET(INPUT);
   ch := INPUT ^
  END;
 WHILE INPUT ^ IN wordchars DO GET(INPUT);
 checkkeywords
END; { proc readword }

PROCEDURE readnumber;

{ reads in a numeric value a character at a time,
  where the sign is optional, and assigns the numeric
  value to token.tnum }

VAR sign : (negative, positive);
    digit,
    fractdiv : INTEGER;

BEGIN
 token.ttype := numeral;
 sign := positive;
 token.tnum := 0;
 IF INPUT ^ IN ['-', '+']
  THEN BEGIN
   IF INPUT ^ = '-'
    THEN sign := negative;
   GET(INPUT)
  END;
 IF NOT (INPUT ^ IN digits)
  THEN token.ttype := errtoken
  ELSE BEGIN
   WHILE INPUT ^ IN digits
    DO BEGIN
     digit := ORD(INPUT ^) - ORD('0');
     token.tnum := token.tnum*10 + digit;
     GET(INPUT)
    END;
   IF INPUT ^ = '.'
    THEN BEGIN
     GET(INPUT);
     fractdiv := 10;
     WHILE INPUT ^ IN digits
      DO BEGIN
       digit := ORD(INPUT ^) - ORD('0');
       token.tnum := token.tnum + digit/fractdiv;
       GET(INPUT);
       fractdiv := fractdiv * 10
      END
    END;
```

```
     IF sign = negative
      THEN token.tnum := token.tnum * (-1);
    END
  END; { proc readnumber }

BEGIN { gettoken }
 skipblankchar;
 IF INPUT ^ IN letters THEN readword
 ELSE IF INPUT ^ IN numberstart THEN readnumber
 ELSE IF EOLN(INPUT) THEN token.ttype := endofline
 ELSE IF INPUT ^ = '='
  THEN BEGIN
   token.ttype := assign;
   GET(INPUT)
  END
 ELSE IF INPUT ^ = '$'
  THEN BEGIN
   token.ttype := endofinfo;
   GET(INPUT)
  END
 ELSE IF INPUT ^ = '.'
  THEN BEGIN
   token.ttype := dot;
   GET(INPUT)
  END
 ELSE token.ttype := errtoken
END; { proc gettoken }


PROCEDURE reporterror (
  errorstate : errortype;
  errorarg : textmessage );

  PROCEDURE trimarg;

  { to remove trailing spaces from errorarg }

  BEGIN
   WHILE (errorarg.length > 1)
         AND (errorarg[errorarg.length] = ' ')
    DO errorarg.length := errorarg.length - 1
  END; { proc trimarg }

BEGIN { reporterror }
 trimarg;
 CASE errorstate of
  attexist  : WRITELN('Error, attributes have already been ',
                      'declared for ',errorarg);
  attdup    : WRITELN('Error, attribute name ',errorarg,
                      ' has been duplicated');
  attmiss   : WRITELN('Error, attribute ',errorarg,
                      ' does not exist');
  dsexists  : WRITELN('Error, data set ',errorarg,
                      ' already exists');
  dsmiss    : WRITELN('Error, data set ',errorarg,
                      ' does not exist');
  entexists : WRITELN('Error, entity type ',errorarg,
                      ' already exists');
  entmiss   : WRITELN('Error, entity type ',errorarg,
                      ' does not exist');
  eqexp     : WRITELN('Error, = expected after ',errorarg);
  identexp  : WRITELN('Error, identifier expected for ',errorarg);
  insuffinst: WRITELN('Error, insufficient instances (<3) ',
                         'have been declared for ',errorarg);
   invarg     : WRITELN('Error, invalid argument found');
   invinfo    : WRITELN('Error, token found is not an ',
                         'appropriate keyword');
   invlevel   : WRITELN('Error, invalid level of measurement given');
   invmeas    : WRITELN('Error, measurement specified is unsuitable');
   invmeastyp: WRITELN('Error, meas type of QUAL or QUANT must ',
                         'be specified');
   invnumarg  : WRITELN('Error, number of arguments expected = ',
                         errorarg);
   invrel     : WRITELN('Error, invalid relationship specified');
   invval     : WRITELN('Error, an invalid data value has been ',
                         'entered for ',errorarg);
   measexists: WRITELN('Error, measurement scheme already exists');
   measmiss   : WRITELN('Error, measurement scheme ',errorarg,
                          ' does not exist');
   noatts     : WRITELN('No attributes have been declared for ',
                         errorarg);
   noinsts    : WRITELN('No instances have been declared for ',
                         errorarg);
   nolevel    : WRITELN('Error, no LEVEL value declared');
   nomeas     : WRITELN('Error, no MEAS value declared');
   notype     : WRITELN('Error, no TYPE value declared');
   numexp     : WRITELN('Error, numeric value expected for ',errorarg)
  END
END; { proc reporterror }


%INCLUDE 'MODEL_ROUTINES.PAS/NOLIST'
%INCLUDE 'CHECK_ROUTINES.PAS/NOLIST'


PROCEDURE setup;

{ initialise knowledge base }

BEGIN
 lowercase := ['a'..'z'];
 uppercase := ['A'..'Z'];
 letters := lowercase + uppercase;
 digits := ['0'..'9'];
 wordchars := letters + digits + [underscore];
 numberstart := ['0'..'9','-','+'];
 explain := TRUE;
 loadknowbase
END; { proc setup }


BEGIN { main program }
 setup;
 REPEAT
  WRITELN;
  WRITE(prompt);
  skipwhitespace;
  gettoken;
  CASE token.ttype OF
   addatt       : atttypeaddition;
   adddds       : dstypeaddition;
   addent       : etypeaddition;
   addinst      : instaddition;
   addmeas      : measaddition;
   showatt      : displayatt;
   showdsdir    : displaydsdir;
```

```
    showentdir  : displayentdir;
    showinst    : displayinst;
    showmeas    : displaymeas;
    showmeasdir : displaymeasdir;
    exptok      : explain := TRUE;
    noexptok    : explain := FALSE;
    association : procstatreq(association, nulltest, twosample);
    location    : procstatreq(location, nulltest, ksample);
    pearson..coeff_of_cont :
                  procstatreq(association, token.ttype, twosample);
    normal_test..fisher_exact :
                  procstatreq(location, token.ttype, ksample);
    quit        : WRITELN('Exiting from program');
    OTHERWISE     WRITELN('Error, invalid command')
   END;
   READLN;
  UNTIL token.ttype = quit
END. { prog main }
```

B.2 Model_routines.pas

```
{ file model_routines.pas }


CONST dir = '[LAWSONKW.PROJECT]';
      attspec = '.ATT';

TYPE  filename = VARYING [doublelength] OF CHAR;

VAR   keyworddir,
      checkdir,
      entdir,
      dsdir,
      attdir,
      datafile,
      measdir,
      convdir : TEXT;

PROCEDURE saveetype (
  VAR ename,
      supername : word;
  VAR super_rel : taxon_relation ); FORWARD;

PROCEDURE savedstype (
  VAR newname,
      newtype : word ); FORWARD;

PROCEDURE saveattlist (
  VAR dsname : word;
      atthead : attnodepointer ); FORWARD;

PROCEDURE savedata (
  VAR ptods : dsnodepointer ); FORWARD;

PROCEDURE savemeasscheme (
  VAR ptomeas : measpointer ); FORWARD;

PROCEDURE saveconvscheme (
  VAR ptoconv : convpointer ); FORWARD;
```

```
{********** pascal structure routines **********}


PROCEDURE etypesearch (
  VAR currentnodep : enodepointer;
  VAR req_ent : word;
  VAR reqnodep : enodepointer );

{ search recursively for entity type req_ent,
  reqnodep is set to point to it if found and
  set to NIL otherwise }

BEGIN
 IF currentnodep = NIL
  THEN reqnodep := NIL
  ELSE WITH currentnodep ^ DO
   IF ent_name = req_ent
    THEN reqnodep := currentnodep
    ELSE BEGIN
     etypesearch(subpointer, req_ent, reqnodep);
     IF reqnodep = NIL
      THEN etypesearch(nextpointer, req_ent, reqnodep)
    END
END; { proc etypesearch }


PROCEDURE addetype (
  VAR super_p : enodepointer;
  VAR new_type : word;
  VAR link_type : taxon_relation );

{ add new entity type new_type to the sub-types
  of super_p with relationship link_type }

VAR temp_p : enodepointer;

BEGIN
 NEW(temp_p);
 WITH temp_p ^
  DO BEGIN
   ent_name := new_type;
   super_rel := link_type;
   superpointer := super_p;
   subpointer := NIL;
   nextpointer := super_p ^.subpointer
  END;
 super_p ^.subpointer := temp_p
END; { proc addetype }


PROCEDURE etypeaddition;

{ input arguments for ADDENT command and check valid,
  if OK, call procedures to add the new type to the taxonomy
  and to the entity directory file }

LABEL endofproc;
VAR  new_ent,
     super_ent : word;
     ent_p : enodepointer;
     taxon_link : taxon_relation;
  PROCEDURE dealwitherror (
    errorstate : errortype;
    errorarg : textmessage );

  BEGIN
   reporterror(errorstate, errorarg);
   GOTO endofproc
  END;

BEGIN
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'new entity type');
 new_ent := token.tchars;
 etypesearch(ent_root, new_ent, ent_p);
 IF ent_p <> NIL
  THEN dealwitherror(entexists, new_ent);
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'super entity type');
 super_ent := token.tchars;
 etypesearch(ent_root, super_ent, ent_p);
 IF ent_p = NIL
  THEN dealwitherror(entmiss, super_ent);
 gettoken;
 CASE token.ttype OF
  nongentok : taxon_link := nongeneric;
  gentok    : taxon_link := generic;
  OTHERWISE   dealwitherror(invrel, nullname)
 END;
 WRITELN;
 addetype(ent_p, new_ent, taxon_link);
 saveetype(new_ent, super_ent, taxon_link);
 endofproc:
END; { proc etypeaddition }


PROCEDURE displayentdir;

{ input argument for the SHOWENTDIR command,
  if OK, call the display procedure to recursively print
  out the entity taxonomy below the given argument }

LABEL endofproc;
VAR entname : word;
    reqp : enodepointer;

  PROCEDURE dealwitherror (
    errorstate : errortype;
    errorarg : textmessage );

  BEGIN
   reporterror(errorstate, errorarg);
   GOTO endofproc
  END;

  PROCEDURE display (
    VAR currentnodep : enodepointer;
        margin : integer );

  BEGIN
   WITH currentnodep ^
    DO BEGIN
```

```
      WRITELN(ent_name:margin, super_rel:50-margin);
      IF subpointer <> NIL
       THEN display(subpointer, margin+2);
      IF nextpointer <> NIL
       THEN display(nextpointer, margin)
     END
   END; { display }

 BEGIN { displayentdir }
  gettoken;
  IF token.ttype <> identifier
   THEN dealwitherror(identexp, 'entity type');
  entname := token.tchars;
  etypesearch(ent_root, entname, reqp);
  IF reqp = NIL
   THEN dealwitherror(entmiss, entname);
  WRITELN; WRITELN;
  WRITELN('Entity type','Relation to super type':39);
  WRITELN;
  WITH reqp ^
   DO BEGIN
    WRITELN(ent_name:wordlength, super_rel:50-wordlength);
    IF subpointer <> NIL
     THEN display(subpointer, wordlength+2)
   END;
  endofproc:
 END; { proc displayentdir }


 PROCEDURE dstypesearch (
   VAR ptocurrent : dsnodepointer;
   VAR reqds : word;
   VAR ptoreqds : dsnodepointer );

 { search recursively for data set reqds,
   ptoreqds is set to point to it if found
   and set to NIL otherwise }

 BEGIN
  IF ptocurrent = NIL
   THEN ptoreqds := NIL
   ELSE WITH ptocurrent ^ DO
    IF reqds = ds_name
     THEN ptoreqds := ptocurrent
     ELSE IF reqds < ds_name
      THEN dstypesearch(leftp, reqds, ptoreqds)
      ELSE dstypesearch(rightp, reqds, ptoreqds)
 END; { proc dstypesearch }


 PROCEDURE  adddstype (
   VAR ptocurrent,
       newnode : dsnodepointer );

 { add the new data set newnode to the alphabetically
   ordered binary tree of data sets }

 BEGIN
  IF ptocurrent = NIL
   THEN ptocurrent := newnode
   ELSE IF newnode ^.ds_name < ptocurrent ^.ds_name
    THEN adddstype(ptocurrent ^.leftp, newnode)
    ELSE adddstype(ptocurrent ^.rightp, newnode)
```

```
END; { proc adddstype }


PROCEDURE dstypeaddition;

{ input arguments for the  ADDDS command,
  if OK, generate a ds_node and call procedures to
  add it to the data set tree and directory file }

LABEL endofproc;
VAR newname, newtype : word;
    ptods : dsnodepointer;

  PROCEDURE dealwitherror (
    errorstate : errortype;
    errorarg : textmessage );

  BEGIN
   reporterror(errorstate, errorarg);
   GOTO endofproc
  END;

BEGIN
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'new data set name');
 newname := token.tchars;
 dstypesearch(ds_root, newname, ptods);
 IF ptods <> NIL THEN dealwitherror(dsexists, newname);
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'entity type of dataset');
 newtype := token.tchars;
 NEW(ptods);
 WITH ptods ^
  DO BEGIN
   ds_name := newname;
   leftp := NIL;
   rightp := NIL;
   instances := 0;
   attchain := NIL;
   etypesearch(ent_root, newtype, ent_type)
  END;
 IF ptods ^.ent_type = NIL
  THEN BEGIN
   DISPOSE(ptods);
   dealwitherror(entmiss, newtype)
  END;
 WRITELN;
 adddstype(ds_root, ptods);
 savedstype(newname, newtype);
 endofproc:
END; { proc dstypeaddition }


PROCEDURE displaydsdir;

{ for SHOWDSDIR command, call procedure display to recursively
  print the data set tree in alphabetical order}

  PROCEDURE display (
    VAR currentnode : ds_node );
   BEGIN
    WITH currentnode
     DO BEGIN
      IF leftp <> NIL
       THEN display(leftp ^);
      WRITELN(ds_name, ent_type ^.ent_name:18, instances:7);
      IF rightp <> NIL
       THEN display(rightp ^)
     END
   END; { proc display }

 BEGIN { displaydsdir }
  IF ds_root = NIL
   THEN WRITELN('No data sets declared')
   ELSE BEGIN
    WRITELN; WRITELN;
    WRITELN('Data set','Entity type':21,'Instances':16);
    WRITELN;
    display(ds_root ^)
   END
 END; { proc displaydsdir }


 PROCEDURE meassearch (
   VAR ptocurrent : measpointer;
   VAR reqmeas : word;
   VAR ptoreqmeas : measpointer );

 { recursively search for scheme reqmeas, set ptoreqmeas
   to point to it if found otherwise set to NIL }

 BEGIN
  IF ptocurrent = NIL
   THEN ptoreqmeas := NIL
   ELSE WITH ptocurrent ^ DO
    IF reqmeas = measname
     THEN ptoreqmeas := ptocurrent
     ELSE IF reqmeas < measname
      THEN meassearch(leftp, reqmeas, ptoreqmeas)
      ELSE meassearch(rightp, reqmeas, ptoreqmeas)
 END; { proc meassearch }


 PROCEDURE addmeasscheme (
   VAR ptomeas,
       newmeas : measpointer );

 { add newmeas to appropriate place in alphabetically
   ordered binary tree of measurement schemes }

 BEGIN
  IF ptomeas = NIL
   THEN ptomeas := newmeas
  ELSE IF newmeas ^.measname < ptomeas ^.measname
   THEN addmeasscheme(ptomeas ^.leftp, newmeas)
   ELSE addmeasscheme(ptomeas ^.rightp, newmeas)
 END; { proc addmeasscheme }


 PROCEDURE catsearch (
   VAR cathead : catnodepointer;
   VAR token : tokeninfo;
   VAR catpos : INTEGER );
```

```
{ search for category value reqcharcat/reqnumcat
  and return index position via catpos which is set
  to 0 if the value is not found }

VAR ptocat : catnodepointer;
    index : INTEGER;

BEGIN
 index := 0;
 catpos := 0;
 ptocat := cathead;
 WHILE (ptocat <> NIL) AND (catpos = 0)
  DO BEGIN
   index := index + 1;
   CASE ptocat ^.cattype OF
    identifier :
     IF token.tchars = ptocat ^.charvalue
      THEN catpos := index
      ELSE ptocat := ptocat ^.next;
    numeral :
     IF token.tnum = ptocat ^.numvalue
      THEN catpos := index
      ELSE ptocat := ptocat ^.next
   END
  END
END; { proc catsearch }


PROCEDURE newcatnode (
  VAR cathead,
      current : catnodepointer );

{ generate new cat_node, place at end of list headed
  by cathead and set current to point to it }

BEGIN
 IF cathead = NIL
  THEN BEGIN
   NEW(cathead);
   current := cathead
  END
  ELSE BEGIN
   NEW(current ^.next);
   current := current ^.next
  END;
 current ^.next := NIL
END; { proc newcatnode }


PROCEDURE getqualclasses (
  VAR ptomeas : measpointer;
      measspec : sortofmeas );

{ input category values for qualitative scheme and call
  procedure newcatnode in building chain of value nodes }

VAR typeofdata : datatype;
    indexnum,
    catindex : INTEGER;
    ptocat : catnodepointer;
    morecats : BOOLEAN;
```

```
BEGIN
 WRITE('Enter values one per line, ');
 IF ptomeas ^.ordtype = ordered
  THEN WRITE('in ascending order, ');
 WRITELN('terminating list with $');
 typeofdata := ptomeas ^.cattype;
 indexnum := 1;
 morecats := TRUE;
 WHILE morecats
  DO BEGIN
   REPEAT
    READLN;
    WRITE('Enter category ',indexnum:3,' : ');
    gettoken;
   UNTIL token.ttype IN [typeofdata, endofinfo];
   WRITELN;
   IF token.ttype = typeofdata
    THEN BEGIN
     catsearch(ptomeas ^.cathead, token, catindex);
     IF catindex <> 0
      THEN WRITELN('This category value has already been entered')
      ELSE BEGIN
       newcatnode(ptomeas ^.cathead, ptocat);
       ptocat ^.cattype := typeofdata;
       CASE typeofdata OF
        identifier : ptocat ^.charvalue := token.tchars;
        numeral    : ptocat ^.numvalue := token.tnum
       END;
       IF (indexnum = 2) AND (measspec IN [orddich, unorddich])
        THEN morecats := FALSE;
       indexnum := indexnum + 1
      END
    END
    ELSE BEGIN
     morecats := indexnum <= 2;
     IF morecats
      THEN BEGIN
       WRITELN('At least 2 categories must be declared');
       IF ptomeas ^.cathead <> NIL
        THEN BEGIN
         DISPOSE(ptomeas ^.cathead);
         ptomeas ^.cathead := NIL
        END;
       indexnum := 1
      END
    END
  END;
 ptomeas ^. numofcat := indexnum - 1
END; { proc getqualclasses }


PROCEDURE getqualinfo (
  VAR newname : word;
      measspec : sortofmeas;
  VAR ptomeas : measpointer );

{ get info about new qualitative meas newname which is of
  type measspec, if a closed set call proc getqualclasses }

BEGIN
 NEW(ptomeas);
 WITH ptomeas ^
  DO BEGIN
```

```
    measname := newname;
    leftp := NIL;
    rightp := NIL;
    meas_type := qualmeas;
    numofcat := 0;
    cathead := NIL;
    IF measspec = qual
     THEN BEGIN
      REPEAT
       READLN;
       WRITE('Is the set of values open/closed : ');
       gettoken;
      UNTIL token.ttype IN [opentok, closedtok];
      WRITELN;
      IF token.ttype = opentok
       THEN settype := openset
       ELSE settype := closedset
     END
     ELSE settype := closedset;
    IF settype = openset
     THEN ordtype := unordered
    ELSE IF measspec IN [unordqual, unorddich]
     THEN ordtype := unordered
    ELSE IF measspec IN [ordqual, orddich]
     THEN ordtype := ordered
    ELSE BEGIN
     REPEAT
      READLN;
      WRITE('Are the values ordered yes/no : ');
      gettoken;
     UNTIL token.ttype IN [yestok, notok];
     WRITELN;
     IF token.ttype = notok
      THEN ordtype := unordered
      ELSE ordtype := ordered
    END;
    REPEAT
     READLN;
     WRITE('Are the data items of type character/numeric : ');
     gettoken;
    UNTIL token.ttype IN [chartok, numtok];
    WRITELN;
    IF token.ttype = chartok
     THEN cattype := identifier
     ELSE cattype := numeral;
    IF settype = closedset
     THEN getqualclasses(ptomeas, measspec)
   END;
  addmeasscheme(meas_root, ptomeas);
  savemeasscheme(ptomeas)
 END; { proc getqualinfo }


 PROCEDURE getquantinfo (
   VAR newname : word;
   VAR ptomeas : measpointer );

 { get info about quantitative scheme }

 BEGIN
  NEW(ptomeas);
  WITH ptomeas ^
   DO BEGIN
```

```
    measname := newname;
    leftp := NIL;
    rightp := NIL;
    meas_type := quantmeas;
    REPEAT
     READLN;
     WRITE('Enter the lower bound of the meas scheme : ');
     gettoken;
    UNTIL (token.ttype = numeral) AND (token.tnum < maxreal);
    WRITELN;
    lowerbound := token.tnum;
    REPEAT
     READLN;
     WRITE('Enter the upper bound of the meas scheme : ');
     gettoken;
    UNTIL (token.ttype = numeral) AND (token.tnum > lowerbound);
    WRITELN;
    upperbound := token.tnum
   END;
  addmeasscheme(meas_root, ptomeas);
  savemeasscheme(ptomeas)
 END; { proc getquantinfo }


 PROCEDURE measaddition;

 { input arguments for ADDMEAS command and check valid,
   if OK, generate new meas_node, get info and add new scheme
   to measurement tree and directory file }

 LABEL endofproc;
 VAR newmeas : word;
     ptomeas : measpointer;

   PROCEDURE dealwitherror (
     errorstate : errortype;
     errorarg : textmessage );

   BEGIN
    reporterror(errorstate, errorarg);
    GOTO endofproc
   END;

 BEGIN
  gettoken;
  IF token.ttype <> identifier
   THEN dealwitherror(identexp, 'new measurement name');
  newmeas := token.tchars;
  meassearch(meas_root, newmeas, ptomeas);
  IF ptomeas <> NIL
   THEN dealwitherror(measexists, newmeas);
  gettoken;
  CASE token.ttype OF
   qualmeas  : getqualinfo(newmeas, qual, ptomeas);
   quantmeas : getquantinfo(newmeas, ptomeas);
   OTHERWISE   dealwitherror(invmeastyp, nullname)
  END;
  endofproc:
 END; { proc measaddition }


 FUNCTION meastypeOK (
   VAR ptomeas : measpointer;
```

```
      measspec : sortofmeas ) : BOOLEAN;

{ see if meas pointed at by ptomeas conforms to measspec }

BEGIN
 WITH ptomeas ^ DO
  CASE measspec OF
   quant     : meastypeOK := meas_type = quantmeas;
   ordqual   : meastypeOK := (meas_type = qualmeas) AND
                             (ordtype = ordered);
   unordqual : meastypeOK := (meas_type = qualmeas) AND
                             (ordtype = unordered);
   qual      : meastypeOK := meas_type = qualmeas;
   orddich   : meastypeOK := (meas_type = qualmeas) AND
                             (ordtype = ordered) AND (numofcat = 2);
   unorddich : meastypeOK := (meas_type = qualmeas) AND
                             (ordtype = unordered) AND
                             (numofcat = 2);
   dich      : meastypeOK := (meas_type = qualmeas) AND
                             (numofcat = 2)
  END
END; { funct meastypeOK }


PROCEDURE displaymeas;

{ input argument for SHOWMEAS command and check valid,
  if OK, output info about required measurement scheme }

LABEL endofproc;
VAR   dispmeas : word;
      ptomeas : measpointer;
      ptocat : catnodepointer;

  PROCEDURE dealwitherror (
    errorstate : errortype;
    errorarg : textmessage );

  BEGIN
   reporterror(errorstate, errorarg);
   GOTO endofproc
  END;

BEGIN
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'measurement name');
 dispmeas := token.tchars;
 meassearch(meas_root, dispmeas, ptomeas);
 IF ptomeas = NIL
  THEN dealwitherror(measmiss, dispmeas);
 WRITELN; WRITELN;
 WITH ptomeas ^
  DO BEGIN
   CASE meas_type OF
    qualmeas :
     BEGIN
      WRITE('  Meas type = qualitative,');
      IF settype = openset
       THEN WRITELN('  Set type = open,  Data type =',cattype:11)
       ELSE BEGIN
        IF ordtype = unordered
         THEN WRITELN('  Categories are unordered')
```

```
          ELSE WRITELN('  Categories are ordered');
         WRITELN;
         WRITELN('  Valid category values are :-');
         ptocat := cathead;
         WHILE ptocat <> NIL
          DO WITH ptocat ^ DO BEGIN
           CASE cattype OF
            identifier : WRITELN(charvalue:20);
            numeral    : WRITELN(numvalue:12:2)
           END;
           ptocat := next
          END
        END
      END;
     quantmeas :
      BEGIN
       WRITELN('  Meas type = quantitative');
       WRITE('  Lower bound =');
       IF lowerbound = minreal
        THEN WRITELN('min':4)
        ELSE WRITELN(lowerbound:12:2);
       WRITE('  Upper bound =');
       IF upperbound = maxreal
        THEN WRITELN('max':4)
        ELSE WRITELN(upperbound:12:2)
      END
    END
   END;
  endofproc:
 END; { proc displaymeas }


 PROCEDURE displaymeasdir;

 { for SHOWMEASDIR command, call procedure display to recursively
   print the measurement schemes in alphabetical order }

   PROCEDURE display (
     VAR currentmeas : meas_node );

   BEGIN
    WITH currentmeas
     DO BEGIN
      IF leftp <> NIL THEN display(leftp ^);
      WRITELN(measname, meas_type:11);
      IF rightp <> NIL THEN display(rightp ^)
     END
   END; { proc display }

 BEGIN { displaymeasdir }
  IF meas_root = NIL
   THEN WRITELN('No measurement schemes defined')
   ELSE BEGIN
    WRITELN; WRITELN;
    WRITELN('Meas name','Meas type':18);
    WRITELN;
    display(meas_root ^)
   END
 END; { proc displaymeasdir }


 PROCEDURE displaycandmeas (
   measspec : sortofmeas );
```

```
{ display those meas schemes which are of type measspec }

VAR index : INTEGER;

  PROCEDURE display (
    VAR ptomeas : measpointer );

  BEGIN
   IF ptomeas ^.leftp <> NIL
    THEN display(ptomeas^. leftp);
   IF meastypeOK(ptomeas, measspec)
    THEN BEGIN
     IF (index MOD 3 = 0) AND (index <> 0)
      THEN WRITELN;
     index := index + 1;
     WRITE(ptomeas ^.measname:18)
    END;
   IF ptomeas^. rightp <> NIL
    THEN display(ptomeas ^.rightp)
  END; { proc display }

BEGIN { displaycandmeas }
 index := 0;
 display(meas_root);
 WRITELN;
 IF index = 0
  THEN WRITELN('No candidate measurement schemes declared')
END; { proc displaycandmeas }


FUNCTION wantstodecmeas : BOOLEAN;

BEGIN
 WRITELN('This meas scheme is new to the system');
 REPEAT
  READLN;
  WRITE('Do you wish to declare it (yes/no) : ');
  gettoken;
 UNTIL token.ttype IN [yestok, notok];
 WRITELN;
 wantstodecmeas := token.ttype = yestok
END; { funct wantstodecmeas }


PROCEDURE convsearch (
  VAR ptocurrent : convpointer;
      reqconv : doubleword;
  VAR ptoreqconv : convpointer );

{ search for reqconv in conversion directory,
  set ptoreqconv to point to it if found otherwise to NIL }

BEGIN
 IF ptocurrent = NIL
  THEN ptoreqconv := NIL
  ELSE WITH ptocurrent ^ DO
   IF reqconv = from_to
    THEN ptoreqconv := ptocurrent
    ELSE IF reqconv < from_to
     THEN convsearch(left_p, reqconv, ptoreqconv)
     ELSE convsearch(right_p, reqconv, ptoreqconv)
END; { proc convsearch }
```

```
PROCEDURE addconvscheme (
  VAR ptocurrent,
      newconv : convpointer );

{ add newconv to the appropriate place in alphabetically
  ordered binary tree of conversion schemes }

BEGIN
 IF ptocurrent = NIL
  THEN ptocurrent := newconv
  ELSE IF newconv ^.from_to < ptocurrent ^.from_to
   THEN addconvscheme(ptocurrent ^.left_p, newconv)
   ELSE addconvscheme(ptocurrent ^.right_p, newconv)
END; { proc addconvscheme }


PROCEDURE genqntqltnode (
  VAR head,
      current : qntqltpointer );

{ add new qntqltnode to list headed by head and
  set current to point to it }

BEGIN
 IF head = NIL
  THEN BEGIN
   NEW(head);
   current := head
  END
  ELSE BEGIN
   NEW(current ^.next);
   current := current ^.next
  END;
 current ^.next := NIL
END; { proc genqntqltnode }


PROCEDURE genqltqltnode (
  VAR head,
      current : qltqltpointer );

{ add new qltqltnode to list headed by head and
  set current to point to it }

BEGIN
 IF head = NIL
  THEN BEGIN
   NEW(head);
   current := head
  END
  ELSE BEGIN
   NEW(current ^.next);
   current := current ^.next
  END;
 current ^.next := NIL
END; { proc genqltqltnode }


PROCEDURE getconvinfo (
  VAR frommeas,
      tomeas : measpointer;
  VAR ptoconv : convpointer );

{ get info about converting frommeas to tomeas
  and add to conversion tree and directory file }

  PROCEDURE getqntqntinfo (
    VAR ptoconv : convpointer );

  { prompt for and input info on how to do quant/quant conversion }

  BEGIN
   WITH ptoconv ^
    DO BEGIN
     typeofconv := qnt_qnt;
     NEW(qntqnt_p);
     WITH qntqnt_p ^
      DO BEGIN
       REPEAT
        READLN;
        WRITE('Enter multiplying factor : ');
        gettoken;
       UNTIL (token.ttype = numeral) AND (token.tnum <> 0);
       WRITELN;
       a := token.tnum;
       REPEAT
        READLN;
        WRITE('Enter constant : ');
        gettoken;
       UNTIL token.ttype = numeral;
       WRITELN;
       c := token.tnum
      END
    END
  END; { proc getqntqntinfo }

  PROCEDURE getqntqltinfo (
    VAR ptoconv : convpointer );

  { prompt for and input info on how to do quant/qual conversion }

  VAR ptoqntqlt : qntqltpointer;
      lower : REAL;
      indexnum : INTEGER;

  BEGIN
   WITH ptoconv ^
    DO BEGIN
     typeofconv := qnt_qlt;
     qntqlt_p := NIL;
     lower := frommeas ^.lowerbound;
     REPEAT
      genqntqltnode(qntqlt_p, ptoqntqlt);
      REPEAT
       READLN;
       WRITE('For the range');
       IF lower = minreal
        THEN WRITE('min')
        ELSE WRITE(lower:12:2);
       WRITE(' to : ');
       gettoken;
       IF token.ttype = upper
        THEN BEGIN
         token.ttype := numeral;
```

```
          token.tnum := frommeas ^.upperbound
         END;
       UNTIL (token.ttype = numeral) AND (token.tnum > lower)
             AND (token.tnum <= frommeas ^.upperbound);
       WRITELN;
       ptoqntqlt ^.upper := token.tnum;
       REPEAT
        READLN;
        WRITE('The value is : ');
        gettoken;
        indexnum := 0;
        IF token.ttype = tomeas ^.cattype
         THEN catsearch(tomeas ^.cathead, token, indexnum);
       UNTIL indexnum <> 0;
       WRITELN;
       ptoqntqlt ^.index := indexnum;
       lower := ptoqntqlt ^.upper
      UNTIL ptoqntqlt ^.upper = frommeas ^.upperbound
     END
   END; { proc getqntqltinfo }

   PROCEDURE getqltqltinfo (
     VAR ptoconv : convpointer );

   { prompt for and input info on how to do qual/qual conversion }

   VAR ptoqltqlt : qltqltpointer;
       ptocat : catnodepointer;
       toindexnum : INTEGER;

   BEGIN
    WITH ptoconv ^
     DO BEGIN
      typeofconv := qlt_qlt;
      qltqlt_p := NIL;
      ptocat := frommeas ^.cathead;
      WHILE ptocat <> NIL
       DO BEGIN
        genqltqltnode(qltqlt_p, ptoqltqlt);
        REPEAT
         READLN;
         WRITE('The value ');
         CASE ptocat ^.cattype OF
          identifier : WRITE(ptocat ^.charvalue);
          numeral    : WRITE(ptocat ^.numvalue:12:2)
         END;
         WRITE(' converts to : ');
         gettoken;
         toindexnum := 0;
         IF token.ttype = ptocat ^.cattype
          THEN catsearch(tomeas ^.cathead, token, toindexnum);
        UNTIL toindexnum <> 0;
        WRITELN;
        ptoqltqlt ^.toindex := toindexnum;
        ptocat := ptocat ^.next
       END
     END
   END; { proc getqltqltinfo }

BEGIN { getconvinfo }
 REPEAT
  READLN;
  WRITELN('Is it possible to convert from ', frommeas ^.measname);
```

```
   WRITELN('to    ':31,tomeas ^.measname);
   WRITE('Answer yes/no : ');
   gettoken;
  UNTIL token.ttype IN [yestok, notok];
  WRITELN;
  IF token.ttype = yestok
   THEN BEGIN
    NEW(ptoconv);
    WITH ptoconv ^
     DO BEGIN
      from_to := frommeas ^.measname + tomeas ^.measname;
      left_p := NIL;
      right_p := NIL;
      IF frommeas ^.meas_type = quantmeas
       THEN BEGIN
        IF tomeas ^.meas_type = quantmeas
         THEN getqntqntinfo(ptoconv)
         ELSE getqntqltinfo(ptoconv)
       END
       ELSE getqltqltinfo(ptoconv)
     END;
    addconvscheme(conv_root, ptoconv);
    saveconvscheme(ptoconv)
   END
END; { proc getconvinfo }


PROCEDURE performconv (
  VAR ptoconv : convpointer;
  VAR fromds,
      tods : numarray;
  VAR noitems : INTEGER );

{ convert the data in fromds using ptoconv to tods }

VAR i : INTEGER;
    ptoqntqlt : qntqltpointer;
    ptoqltqlt : qltqltpointer;
    index : REAL;

BEGIN
 CASE ptoconv ^.typeofconv OF
  qnt_qnt :
   WITH ptoconv ^.qntqnt_p ^
    DO FOR i := 1 TO noitems
     DO tods[i] := a * fromds[i] + c;
  qnt_qlt :
   FOR i := 1 TO noitems
    DO BEGIN
     ptoqntqlt := ptoconv ^.qntqlt_p;
     WHILE fromds[i] > ptoqntqlt ^.upper
      DO ptoqntqlt := ptoqntqlt ^.next;
     tods[i] := ptoqntqlt ^.index
    END;
  qlt_qlt :
   FOR i := 1 TO noitems
    DO BEGIN
     ptoqltqlt := ptoconv ^.qltqlt_p;
     index := 1;
     WHILE index <> fromds[i]
      DO BEGIN
       ptoqltqlt := ptoqltqlt ^.next;
       index := index + 1
      END;
     tods[i] := ptoqltqlt ^.toindex
    END
 END
END; { proc performconv }


PROCEDURE atttypesearch (
  VAR atthead : attnodepointer;
  VAR reqatt : word;
  VAR ptoreqatt : attnodepointer );

{ search for reqatt in the list headed by atthead,
  set ptoreqatt to point to it if found otherwise set to NIL }

VAR found : BOOLEAN;

BEGIN
 ptoreqatt := atthead;
 found := FALSE;
 WHILE (ptoreqatt <> NIL) AND NOT found
  DO IF ptoreqatt ^.att_name = reqatt
   THEN found := TRUE
   ELSE ptoreqatt := ptoreqatt ^.next_att
END; { proc atttypesearch }


PROCEDURE addattnode (
  VAR headatt,
      lastatt,
      newatt : attnodepointer );

{ add newatt to list headed by headatt and set
  lastatt to point to it }

BEGIN
 IF headatt = NIL
  THEN headatt := newatt
  ELSE lastatt ^.next_att := newatt;
 lastatt := newatt;
 lastatt ^.next_att := NIL
END; { addattnode }


PROCEDURE atttypeaddition;

{ prompt user for info required for ADDATT command,
  if all OK build list of newly declared attributes }

LABEL endofattinfo, endofproc;
VAR   dsname,
      newtype,
      newmeas : word;
      newlevel : data_levels;
      normalatt : BOOLEAN;
      ptods : dsnodepointer;
      ptoatt,
      lastatt,
      newatt : attnodepointer;
      ptomeas : measpointer;
      state : (readkey, readother, allread);
      attprompt : VARYING [10] OF CHAR;
      attnum : INTEGER;
      measOK : BOOLEAN;

    PROCEDURE dealwitherror (
      errorstate : errortype;
      errorarg : textmessage );

    BEGIN
     reporterror(errorstate, errorarg);
     GOTO endofproc
    END; { proc dealwitherror }

    PROCEDURE infoerror (
      errorstate : errortype;
      errorarg : textmessage );

    BEGIN
     DISPOSE(newatt);
     reporterror(errorstate, errorarg)
    END; { proc infoerror }

  BEGIN { atttypeaddition }
   gettoken;
   IF token.ttype <> identifier
    THEN dealwitherror(identexp, 'data set name');
   dsname := token.tchars;
   dstypesearch(ds_root, dsname, ptods);
   IF ptods = NIL
    THEN dealwitherror(dsmiss, dsname);
   IF ptods ^.attchain <> NIL
    THEN dealwitherror(attexist, dsname);
   WRITELN('Enter key attributes, one per line, ',
           'terminate list with $');
   state := readkey;
   attprompt := 'Key';
   attnum := 1;
   REPEAT
    READLN;
    WRITE(attprompt,attnum:3,' > ');
    gettoken;
    CASE token.ttype OF
     endofinfo :
      BEGIN
       state := SUCC(state);
       IF state = readother
        THEN BEGIN
         WRITELN('Enter other attributes, terminate list with $');
         attprompt := 'Att'
        END
      END;
     identifier :
      BEGIN
       NEW(newatt);
       WITH newatt ^
        DO BEGIN
         atttypesearch(ptods ^.attchain, token.tchars, ptoatt);
         IF ptoatt <> NIL
          THEN BEGIN
           infoerror(attdup, token.tchars);
           GOTO endofattinfo
          END;
         att_name := token.tchars;
         newtype := nullname;
         newlevel := none;
```

```
newmeas := nullname;
normalatt := FALSE;
gettoken;
WHILE token.ttype <> endofline
 DO BEGIN { read in info about attribute }
  CASE token.ttype OF
   typetok :
    BEGIN
     gettoken;
     IF token.ttype <> assign
      THEN BEGIN
       infoerror(eqexp, 'TYPE');
       GOTO endofattinfo
      END;
     gettoken;
     IF token.ttype <> identifier
      THEN BEGIN
       infoerror(identexp, 'type argument');
       GOTO endofattinfo
      END;
     newtype := token.tchars
    END;
   leveltok :
    BEGIN
     gettoken;
     IF token.ttype <> assign
      THEN BEGIN
       infoerror(eqexp, 'LEVEL');
       GOTO endofattinfo
      END;
     gettoken;
     CASE token.ttype OF
      nomtok   : newlevel := nominal;
      ordtok   : newlevel := ordinal;
      ranktok  : newlevel := rank;
      inttok   : newlevel := interval;
      rattok   : newlevel := ratio;
      OTHERWISE  BEGIN
                  infoerror(invlevel, nullname);
                  GOTO endofattinfo
                 END
     END
    END;
   meastok :
    BEGIN
     gettoken;
     IF token.ttype <> assign
      THEN BEGIN
       infoerror(eqexp, 'MEAS');
       GOTO endofattinfo
      END;
     gettoken;
     IF token.ttype <> identifier
      THEN BEGIN
       infoerror(identexp, 'meas argument');
       GOTO endofattinfo
      END;
     newmeas := token.tchars
    END;
   normtok : normalatt := TRUE;
   OTHERWISE  BEGIN
               infoerror(invinfo, nullname);
               GOTO endofattinfo
```

```
              END
  END;
  gettoken
 END;
{ verify info entered about attribute }
IF state = readkey
 THEN att_role := key
 ELSE att_role := non_key;
IF newtype = nullname
 THEN BEGIN
  infoerror(notype, nullname);
  GOTO endofattinfo
 END;
att_type := newtype;
CASE newlevel OF
 none :
  BEGIN
   infoerror(nolevel, nullname);
   GOTO endofattinfo
  END;
 rank :
  BEGIN
   datalevel := newlevel;
   att_dist := distunknown;
   meas_p := NIL;
   mode := numeral;
   num_p := NIL
  END;
 OTHERWISE
  BEGIN
   IF newmeas = nullname
    THEN BEGIN
     infoerror(nomeas, nullname);
     GOTO endofattinfo
    END;
   meassearch(meas_root, newmeas, ptomeas);
   IF ptomeas = NIL
    THEN BEGIN
     IF wantstodecmeas
      THEN CASE newlevel OF
       nominal : getqualinfo(newmeas, qual, ptomeas);
       ordinal :
        BEGIN
         REPEAT
          READLN;
          WRITE('Is this measurement scheme qual/quant : ');
          gettoken;
         UNTIL token.ttype IN [qualmeas, quantmeas];
         IF token.ttype = qualmeas
          THEN getqualinfo(newmeas, ordqual, ptomeas)
          ELSE getquantinfo(newmeas, ptomeas)
        END;
       interval, ratio : getquantinfo(newmeas, ptomeas)
      END
      ELSE GOTO endofattinfo
    END
    ELSE BEGIN
     CASE newlevel OF
      nominal : measOK := meastypeOK(ptomeas, qual);
      ordinal : measOK := meastypeOK(ptomeas, quant) OR
                          meastypeOK(ptomeas, ordqual);
      interval, ratio : measOK :=  meastypeOK(ptomeas, quant)
     END;
```

```
              IF NOT measOK
               THEN BEGIN
                infoerror(invmeas, nullname);
                GOTO endofattinfo
               END;
             END;
            datalevel := newlevel;
            IF (newlevel IN [interval, ratio]) AND normalatt
             THEN att_dist := normaldist
             ELSE att_dist := distunknown;
            meas_p := ptomeas;
            IF (ptomeas ^.settype = openset)
                AND (ptomeas ^.cattype = identifier)
             THEN BEGIN
              mode := identifier;
              char_p := NIL
             END
             ELSE BEGIN
              mode := numeral;
              num_p := NIL
             END
           END
          END
         END;
        addattnode(ptods ^.attchain, lastatt, newatt);
        attnum := attnum + 1;
        endofattinfo :
       END;
      endofline : ;
      OTHERWISE  WRITELN('Error, invalid symbol found')
     END;
   UNTIL state = allread;
   WRITELN;
   saveattlist(ptods ^.ds_name, ptods ^.attchain);
   endofproc:
  END; { proc atttypeaddition }


  PROCEDURE displayatt;

  { read argument for SHOWATT command, if OK list the
    attributes declared for the required data set }

  LABEL endofproc;
  VAR dsname : word;
      ptods : dsnodepointer;
      ptoatt : attnodepointer;

    PROCEDURE dealwitherror (
      errorstate : errortype;
      errorarg : textmessage );

    BEGIN
     reporterror(errorstate, errorarg);
     GOTO endofproc
    END;

  BEGIN
   gettoken;
   IF token.ttype <> identifier
    THEN dealwitherror(identexp, 'data set name');
   dsname := token.tchars;
   dstypesearch(ds_root, dsname, ptods);
  IF ptods = NIL
   THEN dealwitherror(dsmiss, dsname);
  IF ptods ^.attchain = NIL
   THEN dealwitherror(noatts, dsname);
  WRITELN; WRITELN;
  WRITELN('Att name','Att role':18,'Att type':10,'Level':18,
          'Meas':14);
  WRITELN;
  ptoatt := ptods ^.attchain;
  WHILE ptoatt <> NIL
   DO WITH ptoatt ^ DO BEGIN
    WRITE(att_name,att_role:10,att_type:18,datalevel:10);
    IF att_dist = normaldist
     THEN WRITE('NORMAL':8)
     ELSE WRITE(' ':8);
    IF meas_p <> NIL
     THEN WRITE(meas_p ^.measname:18);
    WRITELN;
    ptoatt := next_att
   END;
 endofproc:
END; { proc displayatt }


PROCEDURE instaddition;

{ input argument for ADDINST command and check valid,
  prompt user for data an instance at a time and validate }

LABEL endofproc;
VAR   dsname : word;
      ptods : dsnodepointer;
      noelements : INTEGER;
      erroroccured : BOOLEAN;
      ptoatt : attnodepointer;

  PROCEDURE dealwitherror (
    errorstate : errortype;
    errorarg : textmessage );

  BEGIN
   reporterror(errorstate, errorarg);
   GOTO endofproc
  END; { proc dealwitherror }

  PROCEDURE getlineofdata;

  { input an item of data for primary version of each attribute }

  LABEL endofdatainput;
  VAR index : INTEGER;

    PROCEDURE dataerror (
      errorstate : errortype;
      errorarg : textmessage );

    BEGIN
     reporterror(errorstate, errorarg);
     erroroccured := TRUE;
     GOTO endofdatainput
    END; { proc dataerror }

  BEGIN { getlineofdata }
  ptoatt := ptods ^.attchain;
  REPEAT
   WITH ptoatt ^ DO BEGIN
    IF datalevel = rank
     THEN BEGIN
      IF token.ttype <> numeral
       THEN dataerror(invval, ptoatt ^.att_name);
      IF token.tnum <= 0
       THEN dataerror(invval, ptoatt ^.att_name);
      num_p ^[noelements + 1] := token.tnum
     END
     ELSE CASE meas_p ^.meas_type OF
      qualmeas :
       BEGIN
        IF token.ttype <> meas_p ^.cattype
         THEN dataerror(invval, ptoatt ^.att_name);
        IF meas_p ^.settype = closedset
         THEN BEGIN
          catsearch(meas_p ^.cathead, token, index);
          IF index = 0
           THEN dataerror(invval, ptoatt ^.att_name)
           ELSE token.tnum := index
         END;
        CASE mode OF
         identifier : char_p ^[noelements+1] := token.tchars;
         numeral    : num_p ^[noelements+1] := token.tnum
        END
       END;
      quantmeas :
       BEGIN
        IF token.ttype <> numeral
         THEN dataerror(invval, ptoatt ^.att_name);
        IF (token.tnum < meas_p ^.lowerbound)
           OR (token.tnum > meas_p ^.upperbound)
         THEN dataerror(invval, ptoatt ^.att_name);
        num_p ^[noelements + 1] := token.tnum
       END
     END;
    ptoatt := next_att;
    gettoken;
   END;
  UNTIL ptoatt = NIL;
  endofdatainput :
 END; { proc getlineofdata }

 PROCEDURE sortinstances;

 { sort instances into key order }

 VAR i,
     newpos : INTEGER;
     state : (equal, cont, found);
     tempword : word;
     tempnum : REAL;

 BEGIN
  newpos := 1;
  state := cont;
  WHILE (newpos <= noelements) AND (state = cont)
   DO BEGIN
    ptoatt := ptods ^.attchain;
    state := equal;
    WHILE (ptoatt <> NIL) AND (state = equal)
```

```
      DO IF ptoatt ^.att_role <> key
        THEN ptoatt := NIL
        ELSE WITH ptoatt ^ DO BEGIN
         CASE mode OF
          identifier :
           IF char_p ^[noelements + 1] < char_p ^[newpos]
            THEN state := found
           ELSE IF char_p ^[noelements + 1] > char_p ^[newpos]
            THEN BEGIN
             state := cont;
             newpos := newpos + 1
            END;
          numeral :
           IF num_p ^[noelements + 1] < num_p ^[newpos]
            THEN state := found
           ELSE IF num_p ^[noelements + 1] > num_p ^[newpos]
            THEN BEGIN
             state := cont;
             newpos := newpos + 1
            END
         END;
         IF state = equal THEN ptoatt := next_att
        END
    END;
   IF state = equal
    THEN BEGIN
     WRITELN('Error, key value is not unique');
     erroroccured := TRUE
    END
    ELSE BEGIN { shuffle down elements }
     ptoatt := ptods ^.attchain;
     WHILE ptoatt <> NIL
      DO WITH ptoatt ^ DO BEGIN
       CASE mode OF
        identifier :
         BEGIN
          tempword := char_p ^[noelements + 1];
          FOR i := (noelements + 1) DOWNTO newpos
           DO char_p ^[i+1] := char_p ^[i];
          char_p ^[newpos] := tempword
         END;
        numeral :
         BEGIN
          tempnum := num_p ^[noelements + 1];
          FOR i := (noelements + 1) DOWNTO newpos
           DO num_p ^[i+1] := num_p ^[i];
          num_p ^[newpos] := tempnum
         END
       END;
       ptoatt := next_att
      END
    END
  END; { proc sortinstances }

BEGIN { instaddition }
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'data set name');
 dsname := token.tchars;
 dstypesearch(ds_root, dsname, ptods);
 IF ptods = NIL
  THEN dealwitherror(dsmiss, dsname);
 WITH ptods ^
```

```
   DO BEGIN
    IF attchain = NIL
     THEN dealwitherror(noatts, dsname);
    IF instances = 0 { need to gen data arrays }
     THEN BEGIN
      ptoatt := attchain;
      WHILE ptoatt <> NIL
       DO WITH ptoatt ^ DO BEGIN
        CASE mode OF
         identifier : NEW(char_p);
         numeral    : NEW(num_p)
        END;
        ptoatt := next_att
       END
     END;
    noelements := instances;
    WRITELN('Enter instances one per line, terminate with $');
    REPEAT
     READLN;
     WRITE('DATA> ');
     gettoken;
     CASE token.ttype OF
      endofinfo, endofline : ;
      OTHERWISE
       BEGIN
        erroroccured := FALSE;
        getlineofdata;
        IF NOT erroroccured AND (attchain ^.att_role = key)
         THEN sortinstances;
        IF NOT erroroccured THEN noelements := noelements + 1
       END
     END;
    UNTIL token.ttype = endofinfo;
    WRITELN;
    IF instances <> noelements
     THEN BEGIN
      instances := noelements;
      savedata(ptods)
     END
   END;
 endofproc :
END; { proc instaddition }


FUNCTION charcatvalue (
   VAR ptomeas : measpointer;
       reqindex : REAL ) : WORD;

{ find character category at position reqindex }

VAR index : INTEGER;
    ptocat : catnodepointer;

BEGIN
 index := 1;
 ptocat := ptomeas ^.cathead;
 WHILE index < reqindex
  DO BEGIN
   ptocat := ptocat ^.next;
   index := index + 1
  END;
 charcatvalue := ptocat ^.charvalue
END; { funct charcatvalue }
```

```
FUNCTION numcatvalue (
  VAR ptomeas : measpointer;
      reqindex : REAL ) : REAL;

{ find numeric category at position reqindex }

VAR index : INTEGER;
    ptocat : catnodepointer;

BEGIN
 index := 1;
 ptocat := ptomeas ^.cathead;
 WHILE index < reqindex
  DO BEGIN
   ptocat := ptocat ^.next;
   index := index + 1
  END;
 numcatvalue := ptocat ^.numvalue
END; { funct numcatvalue }


PROCEDURE displayinst;

{ input argument for SHOWINST command and check that
  instances have been declared, display each instance
  for each attribute }

LABEL endofproc;
VAR   dsname : word;
      i : INTEGER;
      ptods : dsnodepointer;
      ptoatt : attnodepointer;

  PROCEDURE dealwitherror (
   errorstate : errortype;
   errorarg : textmessage );

  BEGIN
   reporterror(errorstate, errorarg);
   GOTO endofproc
  END;

BEGIN
 gettoken;
 IF token.ttype <> identifier
  THEN dealwitherror(identexp, 'data set name');
 dsname := token.tchars;
 dstypesearch(ds_root, dsname, ptods);
 IF ptods = NIL
  THEN dealwitherror(dsmiss, dsname);
 WITH ptods ^
  DO BEGIN
   IF attchain = NIL
    THEN dealwitherror(noatts, dsname);
   IF instances = 0
    THEN dealwitherror(noinsts, dsname);
   WRITELN; WRITELN;
   FOR i := 1 TO instances
    DO BEGIN
     ptoatt := attchain;
     WHILE ptoatt <> NIL
```

```
      DO WITH ptoatt ^ DO BEGIN
       IF datalevel = rank
        THEN WRITE(num_p ^[i]:12:2)
        ELSE CASE meas_p ^.meas_type OF
         qualmeas :
          CASE mode OF
           identifier : WRITE(char_p ^[i] :18);
           numeral :
            IF meas_p ^.settype = openset
             THEN WRITE(num_p ^[i] :9:2)
             ELSE CASE meas_p ^.cattype OF
              identifier :
               WRITE(charcatvalue(meas_p, num_p ^[i]):18);
              numeral :
               WRITE(numcatvalue(meas_p, num_p ^[i]):9:2)
             END
          END;
         quantmeas : WRITE(num_p ^[i] :12:2)
        END;
       ptoatt := next_att
      END;
     WRITELN
    END
  END;
 endofproc:
END; { proc displayinst }


{********** file routines **********}


PROCEDURE loadkeywords;

BEGIN
 OPEN(keyworddir, dir + 'KEYWORDDIR', HISTORY := OLD);
 RESET(keyworddir);
 numofkeywords := 0;
 WHILE NOT EOF(keyworddir)
  DO BEGIN
   numofkeywords := numofkeywords + 1;
   WITH keywordtable[numofkeywords]
    DO READLN(keyworddir, keystr, keytoken)
  END;
 CLOSE(keyworddir)
END; { proc loadkeywords }


PROCEDURE loadsemchecks;

  PROCEDURE genchecknode (
    VAR head,
        current : checkpointer );

  BEGIN
   IF head = NIL
    THEN BEGIN
     NEW(head);
     current := head
    END
    ELSE BEGIN
     NEW(current ^.nextcheck);
     current := current ^.nextcheck
    END;
    current ^.nextcheck := NIL
   END; { proc genchecknode }

   PROCEDURE loadcheckset (
    VAR checkarray : ARRAY[lower..upper:statcomms] OF checkpointer;
        checkfile : filename );

   VAR current : checkpointer;
       index : statcomms;

   BEGIN
    OPEN(checkdir, checkfile, HISTORY := OLD);
    RESET(checkdir);
    WHILE NOT EOF(checkdir)
     DO BEGIN
      READ(checkdir, index);
      checkarray[index] := NIL;
      WHILE NOT EOLN(checkdir)
       DO BEGIN
        genchecknode(checkarray[index], current);
        READ(checkdir, current ^.semcheck)
       END;
      READLN(checkdir)
     END;
    CLOSE(checkdir)
   END; { proc loadcheckset }

BEGIN { loadsemchecks }
 loadcheckset(class_checks, dir + 'CLASSCHECKDIR');
 loadcheckset(assoc_checks, dir + 'ASSOCCHECKDIR');
 loadcheckset(loc_checks, dir + 'LOCCHECKDIR')
END; { proc loadsemchecks }


PROCEDURE loadmeasdir;

VAR ptomeas : measpointer;
    ptocat : catnodepointer;
    i : INTEGER;

BEGIN
 meas_root := NIL;
 OPEN(measdir, 'MEASDIR', HISTORY := OLD, ERROR := CONTINUE);
 IF STATUS(measdir) = 0
  THEN BEGIN
   RESET(measdir);
   WHILE NOT EOF(measdir)
    DO BEGIN
     NEW(ptomeas);
     WITH ptomeas ^
      DO BEGIN
       READ(measdir, measname, meas_type);
       leftp := NIL;
       rightp := NIL;
       CASE meas_type OF
        qualmeas :
         BEGIN
          READLN(measdir, cattype, settype, ordtype, numofcat);
          cathead := NIL;
          FOR i := 1 TO numofcat
           DO BEGIN
            newcatnode(cathead, ptocat);
            ptocat ^.cattype := cattype;
            CASE cattype OF
             identifier : READLN(measdir, ptocat ^.charvalue);
             numeral    : READLN(measdir, ptocat ^.numvalue)
            END
           END
         END;
        quantmeas : READLN(measdir, lowerbound, upperbound)
       END
      END;
     addmeasscheme(meas_root, ptomeas)
    END;
   CLOSE(measdir)
  END
END; { proc loadmeasdir }


PROCEDURE loadconvdir;

VAR ptoconv : convpointer;
    ptoqntqlt : qntqltpointer;
    ptoqltqlt : qltqltpointer;

BEGIN
 conv_root := NIL;
 OPEN(convdir, 'CONVDIR', HISTORY := OLD, ERROR := CONTINUE);
 IF STATUS(convdir) = 0
  THEN BEGIN
   RESET(convdir);
   WHILE NOT EOF(convdir)
    DO BEGIN
     NEW(ptoconv);
     WITH ptoconv ^
      DO BEGIN
       READLN(convdir, from_to, typeofconv);
       left_p := NIL;
       right_p := NIL;
       CASE typeofconv OF
        qnt_qnt :
         BEGIN
          NEW(qntqnt_p);
          WITH qntqnt_p ^ DO READLN(convdir, a, c)
         END;
        qnt_qlt :
         BEGIN
          qntqlt_p := NIL;
          WHILE convdir ^ <> '$'
           DO BEGIN
            genqntqltnode(qntqlt_p, ptoqntqlt);
            WITH ptoqntqlt ^ DO READLN(convdir, upper, index)
           END;
          READLN(convdir)
         END;
        qlt_qlt :
         BEGIN
          qltqlt_p := NIL;
          WHILE convdir ^ <> '$'
           DO BEGIN
            genqltqltnode(qltqlt_p, ptoqltqlt);
            WITH ptoqltqlt ^ DO READLN(convdir, toindex)
           END;
          READLN(convdir)
         END
       END;
```

```
        addconvscheme(conv_root, ptoconv)
       END
     END;
    CLOSE(convdir)
   END
END; { proc loadconvdir }


PROCEDURE loadenttree;

VAR ent_name, super_ent : word;
    super_rel : taxon_relation;
    ptosuper : enodepointer;

BEGIN
 NEW(ent_root);
 WITH ent_root ^
  DO BEGIN
   ent_name := 'ROOT        ';
   super_rel := not_applic;
   superpointer := NIL;
   nextpointer := NIL;
   subpointer := NIL
  END;
 OPEN(entdir, 'ENTDIR', HISTORY := OLD, ERROR := CONTINUE);
 IF STATUS(entdir) = 0
  THEN BEGIN
   RESET(entdir);
   WHILE NOT EOF(entdir)
    DO BEGIN
     READLN(entdir, ent_name, super_ent, super_rel);
     etypesearch(ent_root, super_ent, ptosuper);
     addetype(ptosuper, ent_name, super_rel)
    END;
   CLOSE(entdir)
  END
END; { proc loadenttree }


PROCEDURE loadattlist (
  VAR dsname : word;
  VAR atthead : attnodepointer;
  VAR noelements : INTEGER );

VAR attfile : filename;
    lastatt, currentatt : attnodepointer;
    measname : word;
    ch : char;

  PROCEDURE loadchardata (
    VAR char_p : charpointer;
        datafilename : filename );

  BEGIN
   OPEN(datafile, datafilename,
        HISTORY := OLD, ERROR := CONTINUE);
   IF STATUS(datafile) = 0
    THEN BEGIN
     RESET(datafile);
     noelements := 0;
     NEW(char_p);
     WHILE NOT EOF(datafile)
      DO BEGIN
```

```
        noelements := noelements + 1;
        READLN(datafile, char_p ^[noelements])
       END;
      CLOSE(datafile)
     END
     ELSE char_p := NIL
   END; { proc loadchardata }

   PROCEDURE loadnumdata (
     VAR num_p : numpointer;
         datafilename : filename );

   BEGIN
    OPEN(datafile, datafilename,
         HISTORY := OLD, ERROR := CONTINUE);
    IF STATUS(datafile) = 0
     THEN BEGIN
      RESET(datafile);
      noelements := 0;
      NEW(num_p);
      WHILE NOT EOF(datafile)
       DO BEGIN
        noelements := noelements + 1;
        READLN(datafile, num_p ^[noelements])
       END;
      CLOSE(datafile)
     END
     ELSE num_p := NIL
   END; { proc loadnumdata }

 BEGIN { loadattlist }
  attfile := dsname + attspec;
  OPEN(attdir, attfile, HISTORY := OLD, ERROR := CONTINUE);
  IF STATUS(attdir) = 0
   THEN BEGIN
    RESET(attdir);
    WHILE NOT EOF(attdir)
     DO BEGIN
      new(currentatt);
      WITH currentatt ^
       DO BEGIN
        READ(attdir, att_name, att_role, ch, att_type,
             datalevel, att_dist, mode);
        IF datalevel = rank
         THEN meas_p := NIL
         ELSE BEGIN
          READ(attdir, ch, measname);
          meassearch(meas_root, measname, meas_p)
         END;
        CASE mode OF
         identifier : loadchardata(char_p, dsname + att_name);
         numeral    : loadnumdata(num_p, dsname + att_name)
        END;
        READLN(attdir);
        addattnode(atthead, lastatt, currentatt)
       END
     END;
    CLOSE(attdir)
   END
 END; { proc loadattlist }


 PROCEDURE loaddstree;
```

```
VAR ds_type : word;
    newds : dsnodepointer;

BEGIN
 ds_root := NIL;
 OPEN(dsdir, 'DSDIR', HISTORY := OLD, ERROR := CONTINUE);
 IF STATUS(dsdir) = 0
  THEN BEGIN
   RESET(dsdir);
   WHILE NOT EOF(dsdir)
    DO BEGIN
     NEW(newds);
     WITH newds ^
      DO BEGIN
       READLN(dsdir, ds_name, ds_type);
       etypesearch(ent_root, ds_type, ent_type);
       leftp := NIL;
       rightp := NIL;
       instances := 0;
       attchain := NIL;
       loadattlist(ds_name, attchain, instances)
      END;
     adddstype(ds_root, newds)
    END;
   CLOSE(dsdir)
  END
END; { proc loaddstree }


PROCEDURE loadknowbase;

BEGIN
 loadkeywords;
 loadsemchecks;
 loadmeasdir;
 loadconvdir;
 loadenttree;
 loaddstree
END; { proc loadknowbase }


PROCEDURE saveetype {
  VAR ename,
      supername : word;
  VAR super_rel : taxon_relation };

BEGIN
 OPEN(entdir, 'ENTDIR', HISTORY := UNKNOWN);
 EXTEND(entdir);
 WRITELN(entdir, ename, supername, super_rel:11);
 CLOSE(entdir)
END; { proc saveetype }


PROCEDURE savedstype {
  VAR newname,
      newtype : word };

BEGIN
 OPEN(dsdir, 'DSDIR', HISTORY := UNKNOWN);
 EXTEND(dsdir);
 WRITELN(dsdir, newname, newtype);
```

```
  CLOSE(dsdir)
 END; { proc savedstype }


 PROCEDURE saveattlist {
   VAR dsname : word;
       atthead : attnodepointer };

 VAR attfile : filename;

 BEGIN
  attfile := dsname + attspec;
  OPEN(attdir, attfile, HISTORY := NEW);
  REWRITE(attdir);
  WHILE atthead <> NIL
   DO WITH atthead ^ DO BEGIN
    WRITE(attdir, att_name, att_role, att_type:17,
          datalevel, att_dist:12, mode:11);
    IF meas_p <> NIL
     THEN WRITE(attdir, meas_p ^.measname:17);
    WRITELN(attdir);
    atthead := atthead ^.next_att
   END;
  CLOSE(attdir)
 END; { proc saveattlist }


 PROCEDURE savedata {
   VAR ptods : dsnodepointer };

 VAR datafilename : filename;
     ptoatt : attnodepointer;
     i : INTEGER;

 BEGIN
  WITH ptods ^
   DO BEGIN
    ptoatt := attchain;
    WHILE ptoatt <> NIL
     DO WITH ptoatt ^ DO BEGIN
      datafilename := ds_name + att_name;
      OPEN(datafile, datafilename, HISTORY := UNKNOWN);
      REWRITE(datafile);
      CASE mode OF
       identifier :
        FOR i := 1 TO instances
         DO WRITELN(datafile, char_p ^[i]);
       numeral :
        FOR i := 1 TO instances
         DO WRITELN(datafile, num_p ^[i])
      END;
      ptoatt := next_att;
      CLOSE(datafile)
     END
   END
 END; { proc savedata }


 PROCEDURE savemeasscheme {
   VAR ptomeas : measpointer };

 VAR ptocat : catnodepointer;
     i : INTEGER;
```

```
BEGIN
 OPEN(measdir, 'MEASDIR', HISTORY := UNKNOWN);
 EXTEND(measdir);
 WITH ptomeas ^
  DO BEGIN
   WRITE(measdir, measname, meas_type:10);
   CASE meas_type OF
    qualmeas :
     BEGIN
      WRITELN(measdir, cattype:11, settype:10, ordtype:10,
              numofcat:4);
      ptocat := cathead;
      FOR i := 1 TO numofcat
       DO BEGIN
        CASE cattype OF
         identifier : WRITELN(measdir, ptocat ^.charvalue);
         numeral    : WRITELN(measdir, ptocat ^.numvalue)
        END;
        ptocat := ptocat ^.next
       END
     END;
    quantmeas : WRITELN(measdir, lowerbound, upperbound)
   END
  END;
 CLOSE(measdir)
END; { proc savemeasscheme }


PROCEDURE saveconvscheme {
  VAR ptoconv : convpointer };

VAR ptoqntqlt : qntqltpointer;
    ptoqltqlt : qltqltpointer;

BEGIN
 OPEN(convdir, 'CONVDIR', HISTORY := UNKNOWN);
 EXTEND(convdir);
 WITH ptoconv ^
  DO BEGIN
   WRITELN(convdir, from_to, typeofconv:8);
   CASE typeofconv OF
    qnt_qnt :
     WITH qntqnt_p ^ DO WRITELN(convdir, a, c);
    qnt_qlt :
     BEGIN
      ptoqntqlt := qntqlt_p;
      WHILE ptoqntqlt <> NIL
       DO WITH ptoqntqlt ^ DO BEGIN
        WRITELN(convdir, upper, index:3);
        ptoqntqlt := next
       END;
      WRITELN(convdir, '$')
     END;
    qlt_qlt :
     BEGIN
      ptoqltqlt := qltqlt_p;
      WHILE ptoqltqlt <> NIL
       DO WITH ptoqltqlt ^ DO BEGIN
        WRITELN(convdir, toindex:3);
        ptoqltqlt := next
       END;
      WRITELN(convdir, '$')
     END
    END
   END;
 CLOSE(convdir)
END; { proc saveconvscheme }


{ file model_routines.pas }
```

B.3 Check_routines.pas

```
{ file check_routines.pas }


TYPE listheadpointer = ^ listheadnode;
     itempointer = ^ itemnode;
     listheadnode = RECORD
      nexthead : listheadpointer;
      no_items : INTEGER;
      itemhead : itempointer
     END;
     itemnode = RECORD
      dsinfo : dsnodepointer;
      attinfo : attnodepointer;
      convdata : numpointer;
      measinfo : measpointer;
      nextitem : itempointer
     END;
     gpinfopointer = ^ groupinfo;
     gpnodepointer = ^ groupnode;
     groupinfo = RECORD
      measused : measpointer;
      numdivisions : INTEGER;
      grouphead : gpnodepointer
     END;
     groupnode = RECORD
      members : SET OF 1..100;
      freq : INTEGER;
      nextnode : gpnodepointer
     END;
     summarystate = (startstate, allrat, intrat, allqnt, rankqnt,
       rankqlt, ordqnt, allord, nomqnt, allqlt, novalidstate );
     relstate = (relunknown, related, unrelated);
     normalstate = (normunknown, normalOK, assnormal, nonnormal);
     varstate = (varunknown, eqvarOK, asseqvar, uneqvar);
     inststate = (instunknown, instOK, insttoolow);
     freqstate = (frequnknown, freqOK, freqtoolow);
     datastate = (dataunknown, origOK, convOK, cannotconv);

VAR listheadhead : listheadpointer;
    argsummary : summarystate;
    argsrel : relstate;
    argsnormal : normalstate;
    argvar : varstate;
    numinst : inststate;
    argfreq : freqstate;
    controw,
    contcolumn : gpinfopointer;
    conttotal : INTEGER;
```

```
     dichdata,
     qltdata,
     qntdata : datastate;
     shapwilkcoeff : TEXT;


FUNCTION g01bbf (
   VAR i1,
       i2 : INTEGER;
   VAR a : REAL;
   VAR ifail : INTEGER ) : REAL; EXTERN;
{ NAG library routine to return F dist probability }


FUNCTION g01bcf (
   VAR x : REAL;
   VAR n,
       ifail : INTEGER ) : REAL; EXTERN;
{ NAG library routine to return Chi square probability }


PROCEDURE displayarg (
   VAR ptoitem : itempointer );

BEGIN
 WITH ptoitem ^
  DO WRITE(dsinfo ^.ds_name:strlen(dsinfo ^.ds_name), '.',
           attinfo ^.att_name:strlen(attinfo ^.att_name))
END; { proc displayarg }


PROCEDURE displayarglist (
   VAR ptolisthead : listheadpointer );

{ display each argument name in the list headed by ptolisthead }

VAR ptoitem : itempointer;

BEGIN
 ptoitem := ptolisthead ^.itemhead;
 WHILE ptoitem <> NIL
  DO BEGIN
   displayarg(ptoitem);
   WRITELN;
   ptoitem := ptoitem ^.nextitem
  END
END; { proc displayarglist }


PROCEDURE displayargmeas (
   VAR ptolisthead : listheadpointer );

{ display each argument name and the meas used
  for each item in the list headed by list head }

VAR ptoitem : itempointer;
    arglen : INTEGER;

BEGIN
 ptoitem := ptolisthead ^.itemhead;
 WHILE ptoitem <> NIL
  DO WITH ptoitem ^ DO BEGIN
   displayarg(ptoitem);
```

```
    arglen := strlen(dsinfo^.ds_name) + strlen(attinfo^.att_name);
    WRITELN(attinfo ^.meas_p ^.measname:51-arglen);
    ptoitem := nextitem
   END
 END; { proc displayargmeas }


 {********** semantic check routines **********}


 PROCEDURE checkeqdom (
   VAR item1,
       item2 : itempointer;
   VAR checkOK : BOOLEAN );

 { class level check to see if item1 and item2 have
   the same att_type values, assign result to checkOK }

 BEGIN
  checkOK := item1 ^.attinfo ^.att_type
           = item2 ^.attinfo ^.att_type
 END; { proc checkeqdom }


 PROCEDURE checkenttype (
   VAR item1,
       item2 : itempointer;
   VAR checkOK : BOOLEAN );

 { see if item1 and item2 are of similar entity
   types, assign result to checkOK }

   FUNCTION mostgenent (
     ptoent : enodepointer ) : word;

   BEGIN
    WHILE (ptoent ^.super_rel = generic)
      AND (ptoent ^.ent_name <> 'ROOT')
     DO ptoent := ptoent ^.superpointer;
    mostgenent := ptoent ^.ent_name
   END; { funct mostgenent }

 BEGIN { checkenttype }
  IF item1 ^.dsinfo = item2 ^.dsinfo
   THEN checkOK := TRUE
   ELSE checkOK := mostgenent(item1 ^.dsinfo ^.ent_type)
                 = mostgenent(item2 ^.dsinfo ^.ent_type)
 END; { proc checkenttype }


 PROCEDURE checkrelinsts (
   VAR ds1,
       ds2 : dsnodepointer;
   VAR checkOK : BOOLEAN );

 { see if instances in ds1 and ds2 are the same,
   assign result to checkOK }

 TYPE statustype = (nilatt, otheratt, keyatt);
      statetype = (checking, diffkey, samekey);
 VAR  att1, att2 : attnodepointer;
      keystate : statetype;
      att1status, att2status : statustype;
   FUNCTION attstatus (
     VAR ptoatt : attnodepointer ) : statustype;

   BEGIN
    IF ptoatt = NIL
     THEN attstatus := nilatt
     ELSE IF ptoatt ^.att_role = non_key
      THEN attstatus := otheratt
      ELSE attstatus := keyatt
   END; { funct attstatus }

   PROCEDURE comparechar (
     VAR set1,
         set2 : charpointer;
     VAR instances : INTEGER;
     VAR state : statetype );

   VAR index : INTEGER;

   BEGIN
    index := 1;
    WHILE (state = checking) AND (index <= instances)
     DO IF set1 ^[index] <> set2 ^[index]
         THEN state := diffkey
         ELSE index := index + 1
   END; { proc comparechar }

   PROCEDURE comparenum (
     VAR set1,
         set2 : numpointer;
     VAR instances : INTEGER;
     VAR state : statetype );

   VAR index : INTEGER;

   BEGIN
    index := 1;
    WHILE (state = checking) AND (index <= instances)
     DO IF set1 ^[index] <> set2 ^[index]
         THEN state := diffkey
         ELSE index := index + 1
   END; { proc comparenum }

BEGIN { checkrelinsts }
 IF ds1 = ds2
  THEN checkOK := TRUE
  ELSE IF ds1 ^.instances <> ds2 ^.instances
   THEN checkOK := FALSE
   ELSE BEGIN
    att1 := ds1 ^.attchain;
    att2 := ds2 ^.attchain;
    IF (att1 ^.att_role <> key) OR (att2 ^.att_role <> key)
     THEN checkOK := FALSE
     ELSE BEGIN
      keystate := checking;
      WHILE keystate = checking
       DO BEGIN
        att1status := attstatus(att1);
        att2status := attstatus(att2);
        IF (att1status = keyatt) AND (att2status = keyatt)
         THEN BEGIN
          IF att1 ^.att_type <> att2 ^.att_type
```

```
            THEN keystate := diffkey
            ELSE BEGIN
             IF att1 ^.meas_p <> att2 ^.meas_p
              THEN keystate := diffkey
              ELSE BEGIN
               CASE att1 ^.mode OF
                identifier :
                 comparechar(att1 ^.char_p, att2 ^.char_p,
                             ds1 ^.instances, keystate);
                numeral :
                 comparenum(att1 ^.num_p, att2 ^.num_p,
                            ds1 ^.instances, keystate)
               END;
               IF keystate = checking
                THEN BEGIN
                 att1 := att1 ^.next_att;
                 att2 := att2 ^.next_att
                END
              END
            END
          END
         ELSE IF (att1status <> keyatt) AND (att2status <> keyatt)
          THEN keystate := samekey
         ELSE keystate := diffkey
        END;
       checkOK := keystate = samekey
      END
    END
END; { proc checkrelinsts }


PROCEDURE checkrelargs (
  VAR ptolisthead : listheadpointer);

VAR firstitem,
    currentitem : itempointer;
    checkOK : BOOLEAN;

BEGIN
 argsrel := related;
 firstitem := ptolisthead ^.itemhead;
 currentitem := firstitem ^.nextitem;
 WHILE (currentitem <> NIL) AND (argsrel = related)
  DO BEGIN
   checkrelinsts(firstitem ^.dsinfo, currentitem ^.dsinfo, checkOK);
   IF NOT checkOK
    THEN argsrel := unrelated
    ELSE currentitem := currentitem ^.nextitem
  END
END; { proc checkrelargs }


PROCEDURE checknumargs (
      typeoftestargs : testtype;
  VAR numargs : INTEGER;
  VAR checkOK : BOOLEAN );

BEGIN
 CASE typeoftestargs OF
  twosample : checkOK := numargs = 2;
  ksample   : checkOK := numargs >= 2
 END
END; { proc checknumargs }
```

```
PROCEDURE checknormalargs (
  VAR ptolisthead : listheadpointer);

VAR i, j : INTEGER;
    a : ARRAY [3..29, 1..15] OF REAL;
    critw : ARRAY [3..29] OF REAL;
    ptoitem : itempointer;
    checkOK : BOOLEAN;

  PROCEDURE shapwilk (
        y : numarray;
    VAR n : INTEGER;
    VAR checkOK : BOOLEAN );

  VAR k, i, minindex : INTEGER;
      temp,
      ysum,
      ysqrsum,
      Ssqr,
      b,
      w : REAL;

  BEGIN
   checkOK := TRUE;
   FOR k := 1 TO n-1
    DO BEGIN
     minindex := k;
     FOR i := k+1 TO n
      DO IF y[i] < y[minindex]
          THEN minindex := i;
     IF minindex <> k
      THEN BEGIN
       temp := y[k];
       y[k] := y[minindex];
       y[minindex] := temp
      END
    END;
   ysum := 0;
   ysqrsum := 0;
   FOR i := 1 TO n
    DO BEGIN
     ysum := ysum + y[i];
     ysqrsum := ysqrsum + y[i] * y[i]
    END;
   Ssqr := ysqrsum - ysum * ysum / n;
   b := 0;
   FOR i := 1 TO n DIV 2
    DO b := b + a[n, i] * (y[n-i+1] - y[i]);
   w := b*b / Ssqr;
   checkOK := w >= critw[n]
  END; { proc shapwilk }

BEGIN { checknormalargs }
 OPEN(shapwilkcoeff, dir + 'SHAPWILKCOEFF.DAT', HISTORY := OLD);
 RESET(shapwilkcoeff);
 FOR i := 3 TO 29
  DO FOR j := 1 TO (i+1) DIV 2
   DO READ(shapwilkcoeff, a[i,j]);
 { a[i,j] contains coefficient a(n-i+1) for j=n }
 FOR i := 3 TO 29
  DO READ(shapwilkcoeff, critw[i]);
 CLOSE(shapwilkcoeff);
 argsnormal := normalOK;
```

```
  ptoitem := ptolisthead ^.itemhead;
  WHILE (ptoitem <> NIL) AND (argsnormal <> nonnormal)
   DO WITH ptoitem ^ DO BEGIN
    IF (attinfo ^.att_dist <> normaldist)
       AND (dsinfo ^.instances < 30)
     THEN BEGIN
      shapwilk(attinfo ^.num_p ^, dsinfo ^.instances, checkOK);
      IF NOT checkOK
       THEN BEGIN
        WRITELN;
        REPEAT
         READLN;
         WRITE('Do you wish to assume that the data in ');
         displayarg(ptoitem);
         WRITELN;
         WRITE('  is normally distributed (yes/no/default) : ');
         gettoken;
        UNTIL token.ttype IN [yestok, notok, deftok];
        WRITELN;
        CASE token.ttype OF
         yestok        : argsnormal := assnormal;
         notok, deftok : argsnormal := nonnormal
        END
       END
     END;
    ptoitem := nextitem
   END
END; { proc checknormalargs }


PROCEDURE Ftest (
  VAR ptolisthead : listheadpointer;
  VAR checkOK : BOOLEAN );

VAR ptoitem : itempointer;
    v1, v2, icode : INTEGER;
    s1, s2, F, critF : REAL;

  PROCEDURE calcssqr (
    VAR ssqr : REAL;
    VAR df : INTEGER );

  VAR i : INTEGER;
      x, xsqr : REAL;
      ptodata : numpointer;

  BEGIN
   IF ptoitem ^.convdata <> NIL
    THEN ptodata := ptoitem ^.convdata
    ELSE ptodata := ptoitem ^.attinfo ^.num_p;
   x := 0;
   xsqr := 0;
   df := ptoitem ^.dsinfo ^.instances - 1;
   FOR i := 1 TO df+1
    DO BEGIN
     x := x + ptodata ^[i];
     xsqr := xsqr + SQR(ptodata ^[i])
    END;
   ssqr := xsqr/df - SQR(x)/((df+1)*df)
  END; { proc calcssqr }

BEGIN { Ftest }
 ptoitem := ptolisthead ^.itemhead;
```

```
  calcssqr(s1, v1);
  ptoitem := ptoitem ^.nextitem;
  calcssqr(s2, v2);
  icode := 0;
  IF s1 > s2
   THEN BEGIN
    F := s1/s2;
    critF := g01bbf(v1, v2, F, icode)
   END
   ELSE BEGIN
    F := s2/s1;
    critF := g01bbf(v2, v1, F, icode)
   END;
  checkOK := critF >= 0.05
 END; { proc Ftest }


 PROCEDURE Bartlett (
   VAR ptolisthead : listheadpointer;
   VAR checkOK : BOOLEAN );

 VAR ptoitem : itempointer;
     ptodata : numpointer;
     i, k, ni, phi, df, icode : INTEGER;
     ssqrphi, invphi, philnssqr, x, xsqr, ssqr, avssqr,
     A, M, teststat, critchi : REAL;

 BEGIN
  k := ptolisthead ^.no_items;
  ssqrphi := 0;
  phi := 0;
  invphi := 0;
  philnssqr := 0;
  ptoitem := ptolisthead ^.itemhead;
  WHILE ptoitem <> NIL
   DO BEGIN
    ni := ptoitem ^.dsinfo ^.instances;
    IF ptoitem ^.convdata <> NIL
     THEN ptodata := ptoitem ^.convdata
     ELSE ptodata := ptoitem ^.attinfo ^.num_p;
    x := 0;
    xsqr := 0;
    FOR i := 1 TO ni
     DO BEGIN
      x := x + ptodata ^[i];
      xsqr := xsqr + SQR(ptodata ^[i])
     END;
    ssqr := xsqr/(ni-1) - SQR(x)/(ni*(ni-1));
    ssqrphi := ssqrphi + ssqr*(ni-1);
    phi := phi + ni - 1;
    invphi := invphi + 1/(ni-1);
    philnssqr := philnssqr + (ni-1)*LN(ssqr);
    ptoitem := ptoitem ^.nextitem
   END;
  avssqr := ssqrphi / phi;
  M := phi*LN(avssqr) - philnssqr;
  A := (invphi - 1/phi) / (3*(k-1));
  teststat := M / (1+A);
  df := k-1;
  icode := 0;
  critchi := g01bcf(teststat, df, icode);
  checkOK := critchi >= 0.05
 END; { proc Bartlett }
```

```
PROCEDURE Box (
  VAR ptolisthead : listheadpointer;
  VAR checkOK : BOOLEAN );

VAR ptoitem : itempointer;
    ptodata : numpointer;
    k, phi, N, ni, i, v, df, icode : INTEGER;
    ssqrphi, philnssqr, esqr, equad, x, xsqr, xcub, xquad,
    xi, ssqr, avssqr, M, rhosqr, gamma2, teststat : REAL;
    critchi : REAL;

BEGIN
 k := ptolisthead ^.no_items;
 ssqrphi := 0;
 phi := 0;
 philnssqr := 0;
 N := 0;
 esqr := 0;
 equad := 0;
 ptoitem := ptolisthead ^.itemhead;
 WHILE ptoitem <> NIL
  DO BEGIN
   ni := ptoitem ^.dsinfo ^.instances;
   IF ptoitem ^.convdata <> NIL
    THEN ptodata := ptoitem ^.convdata
    ELSE ptodata := ptoitem ^.attinfo ^.num_p;
   x := 0;
   xsqr := 0;
   xcub := 0;
   xquad := 0;
   FOR i := 1 TO ni
    DO BEGIN
     xi := ptodata ^[i];
     x := x + xi;
     xsqr := xsqr + SQR(xi);
     xcub := xcub + xi*SQR(xi);
     xquad := xquad + SQR(SQR(xi))
    END;
   ssqr := xsqr/(ni-1) - SQR(x)/(ni*(ni-1));
   ssqrphi := ssqrphi + ssqr*(ni-1);
   phi := phi + ni - 1;
   philnssqr := philnssqr + (ni-1)*LN(ssqr);
   N := N + ni;
   esqr := esqr + xsqr - SQR(x)/ni;
   equad := equad + xquad - 4*xcub*x/ni + 6*xsqr*SQR(x)/SQR(ni)
            - 3*SQR(SQR(x))/(ni*SQR(ni));
   ptoitem := ptoitem ^.nextitem
  END;
 avssqr := ssqrphi / phi;
 M := phi*LN(avssqr) - philnssqr;
 v := N - k;
 rhosqr := k / (v*(N-1));
 gamma2 := N*SQR(N) * (((v+2)*equad)/(v*SQR(esqr))-3/N) /
           (v*(v+2)*(1+(N-1)*SQR(rhosqr))-3*N);
 teststat := M / (1 + gamma2/2);
 df := k - 1;
 icode := 0;
 critchi := g01bcf(teststat, df, icode);
 checkOK := critchi >= 0.05
END; { proc Box }
```

```
PROCEDURE checkeqvar (
  VAR ptolisthead : listheadpointer );

VAR checkOK : BOOLEAN;

BEGIN
 IF ptolisthead ^.no_items = 2
  THEN Ftest(ptolisthead, checkOK)
 ELSE IF argsnormal = normalOK
  THEN Bartlett(ptolisthead, checkOK)
  ELSE Box(ptolisthead, checkOK);
 IF checkOK
  THEN argvar := eqvarOK
  ELSE BEGIN
   WRITELN;
   REPEAT
    READLN;
    WRITELN('Do you wish to assume that the sample');
    WRITE('  variances are equal (yes/no/default) : ');
    gettoken;
   UNTIL token.ttype IN [yestok, notok, deftok];
   WRITELN;
   CASE token.ttype OF
    yestok         : argvar := asseqvar;
    notok, deftok : argvar := uneqvar
   END
  END
END; { proc checkeqvar }


PROCEDURE checknige30 (
  VAR ptolisthead : listheadpointer );

VAR ptoitem : itempointer;

BEGIN
 numinst := instOK;
 ptoitem := ptolisthead ^.itemhead;
 WHILE (ptoitem <> NIL) AND (numinst = instOK)
  DO WITH ptoitem ^
   DO IF dsinfo ^.instances < 30
       THEN numinst := insttoolow
       ELSE ptoitem := nextitem
END; { proc checknige30 }


PROCEDURE addcontnode (
  VAR infohead : gpinfopointer;
  VAR ptogroup : gpnodepointer );

BEGIN
 IF infohead ^.grouphead = NIL
  THEN BEGIN
   NEW(infohead ^.grouphead);
   ptogroup := infohead ^.grouphead
  END
  ELSE BEGIN
   NEW(ptogroup ^.nextnode);
   ptogroup := ptogroup ^.nextnode
  END;
 ptogroup ^.nextnode := NIL
END; { proc addcontnode }
```

```
PROCEDURE setupcontnodes (
  VAR infohead : gpinfopointer;
  VAR ptoitem : itempointer );

VAR ptogroup : gpnodepointer;
    i : INTEGER;

BEGIN
 NEW(infohead);
 IF ptoitem ^.measinfo <> NIL
  THEN infohead ^.measused := ptoitem ^.measinfo
  ELSE infohead ^.measused := ptoitem ^.attinfo ^.meas_p;
 infohead ^.numdivisions := infohead ^.measused ^.numofcat;
 infohead ^.grouphead := NIL;
 FOR i := 1 TO infohead ^.numdivisions
  DO BEGIN
   addcontnode(infohead, ptogroup);
   WITH ptogroup ^
    DO BEGIN
     members := [i];
     freq := 0;
     nextnode := NIL
    END
  END
END; { proc setupcontnodes }


PROCEDURE setcontfreqs (
  VAR infohead : gpinfopointer;
  VAR ptoitem : itempointer );

VAR ptodata : numpointer;
    ptogroup : gpnodepointer;
    instances,
    i, j : INTEGER;

BEGIN
 IF ptoitem ^.convdata <> NIL
  THEN ptodata := ptoitem ^.convdata
  ELSE ptodata := ptoitem ^.attinfo ^.num_p;
 instances := ptoitem ^.dsinfo ^.instances;
 FOR i := 1 TO instances
  DO BEGIN
   ptogroup := infohead ^.grouphead;
   FOR j := 1 TO TRUNC(ptodata ^[i]) - 1
    DO ptogroup := ptogroup ^.nextnode;
   ptogroup ^.freq := ptogroup ^.freq + 1
  END
END; { proc setcontfreqs }


PROCEDURE formassoccont (
  VAR ptolisthead : listheadpointer );

VAR ptoitem : itempointer;

BEGIN
 ptoitem := ptolisthead ^.itemhead;
 setupcontnodes(controw, ptoitem);
 setcontfreqs(controw, ptoitem);
 ptoitem := ptoitem ^.nextitem;
```

```
  setupcontnodes(contcolumn, ptoitem);
  setcontfreqs(contcolumn, ptoitem);
  conttotal := ptoitem ^.dsinfo ^.instances
 END; { proc formassoccont }


 PROCEDURE formloccont (
   VAR ptolisthead : listheadpointer );

 VAR ptorownode,
     ptocolumnnode : gpnodepointer;
     ptoitem : itempointer;

 BEGIN
  ptoitem := ptolisthead ^.itemhead;
  { set up a row node for each meas category }
  setupcontnodes(controw, ptoitem);
  NEW(contcolumn);
  contcolumn ^.measused := NIL;
  contcolumn ^.numdivisions := ptolisthead ^.no_items;
  contcolumn ^.grouphead := NIL;
  WHILE ptoitem <> NIL
   DO BEGIN
    { add new column node for current arg and update
      the required row frequencies with its data }
    addcontnode(contcolumn, ptocolumnnode);
    setcontfreqs(controw, ptoitem);
    ptocolumnnode ^.freq := ptoitem ^.dsinfo ^.instances;
    conttotal := conttotal + ptocolumnnode ^.freq;
    ptoitem := ptoitem ^.nextitem
   END
 END; { proc formloccont }


 FUNCTION efreqOK : BOOLEAN;

 VAR below5,
     below1,
     totalcells : INTEGER;
     expfreq : REAL;
     ptorownode,
     ptocolumnnode : gpnodepointer;

 BEGIN
  below5 := 0;
  below1 := 0;
  ptorownode := controw ^.grouphead;
  WHILE ptorownode <> NIL
   DO BEGIN
    ptocolumnnode := contcolumn ^.grouphead;
    WHILE ptocolumnnode <> NIL
     DO BEGIN
      expfreq := ptorownode ^.freq * ptocolumnnode ^.freq
                 / conttotal;
      IF expfreq < 5 THEN below5 := below5 + 1;
      IF expfreq < 1 THEN below1 := below1 + 1;
      ptocolumnnode := ptocolumnnode ^.nextnode
     END;
    ptorownode := ptorownode ^.nextnode
   END;
  totalcells := controw ^.numdivisions * contcolumn ^.numdivisions;
  efreqOK := (below5/totalcells < 0.2) AND (below1 = 0)
 END; { funct efreqOK }
```

```
PROCEDURE findminfreq (
  VAR infohead,
      mininfo : gpinfopointer;
  VAR minfreq,
      minindex : INTEGER );

VAR ptogroup : gpnodepointer;
    index : INTEGER;

BEGIN
 IF (infohead ^.measused <> NIL) AND (infohead ^.numdivisions > 2)
  THEN BEGIN
   ptogroup := infohead ^.grouphead;
   index := 1;
   WHILE ptogroup <> NIL
    DO WITH ptogroup ^ DO BEGIN
     IF ((freq = minfreq) AND
         (infohead ^.numdivisions > mininfo ^.numdivisions))
       OR (freq < minfreq)
      THEN BEGIN
       mininfo := infohead;
       minfreq := freq;
       minindex := index
      END;
     ptogroup := nextnode;
     index := index + 1
    END
  END
END; { proc findminfreq }


PROCEDURE combgroups (
  VAR infohead : gpinfopointer;
  VAR toindex,
      fromindex : INTEGER );

VAR ptogroup1,
    ptogroup2,
    ptogroup3 : gpnodepointer;
    i : INTEGER;

BEGIN
 ptogroup1 := infohead ^.grouphead;
 FOR i := 2 TO toindex
  DO ptogroup1 := ptogroup1 ^.nextnode;
 ptogroup2 := ptogroup1;
 FOR i := toindex+1 TO fromindex-1
  DO ptogroup2 := ptogroup2 ^.nextnode;
 ptogroup3 := ptogroup2 ^.nextnode;
 { ptogroup1 points to group at toindex,
   ptogroup2 points to group before fromindex,
   ptogroup3 points to group at fromindex }
 infohead ^.numdivisions := infohead ^.numdivisions - 1;
 WITH ptogroup1 ^
  DO BEGIN
   members := members + ptogroup3 ^.members;
   freq := freq + ptogroup3 ^.freq
  END;
 ptogroup2 ^.nextnode := ptogroup3 ^.nextnode;
 DISPOSE(ptogroup3)
END; { proc combgroups }
```

```
PROCEDURE trytocombgroups (
  VAR datacomb : BOOLEAN );

VAR minfreq, i, j, numcat,
    numonline,
    minindex,
    toindex : INTEGER;
    mininfo : gpinfopointer;
    ptogroup : gpnodepointer;
    ptomeas : measpointer;

BEGIN
 minfreq := MAXINT;
 findminfreq(controw, mininfo, minfreq, minindex);
 findminfreq(contcolumn, mininfo, minfreq, minindex);
 IF minfreq = maxint
  THEN argfreq := freqtoolow { no possibility of combining groups }
  ELSE BEGIN
   ptomeas := mininfo ^.measused;
   IF NOT datacomb
    THEN BEGIN
     WRITELN;
     WRITELN('Frequencies are low, need to combine categories');
     WRITELN('for a qualitative test to be used')
    END;
   WRITELN;
   WRITELN('Categories in ',
           ptomeas ^.measname:strlen(ptomeas ^.measname),
           ' have currently been grouped as follows :-');
   WRITELN('Group  Members','Freq':49);
   i := 1;
   ptogroup := mininfo ^.grouphead;
   numcat := ptomeas ^.numofcat;
   WHILE ptogroup <> NIL
    DO WITH ptogroup ^ DO BEGIN
     WRITE(i:4,' ');
     numonline := 0;
     FOR j := 1 TO numcat
      DO IF j IN members
       THEN BEGIN
        IF numonline = 3
         THEN BEGIN
          WRITELN;
          WRITE('     ');
          numonline := 0
         END;
        CASE ptomeas ^.cattype OF
         identifier : WRITE(charcatvalue(ptomeas, j):18);
         numeral    : WRITE(numcatvalue(ptomeas, j):9:2)
        END;
        numonline := numonline + 1
       END;
     WRITELN(freq:58-numonline*18);
     i := i + 1;
     ptogroup := nextnode
    END;
   WRITE('Need to combine those in group',minindex:3);
   IF ptomeas ^.ordtype = unordered
    THEN WRITELN(' with another group')
    ELSE WRITELN(' with an adjacent group');
   toindex := 0;
   REPEAT
    READLN;
    WRITE('Enter group number to combine with (or NONE) : ');
    gettoken;
    CASE token.ttype OF
     nonetok :
      BEGIN
       { user indicates that categories can be combined no more }
       argfreq := freqtoolow;
       dichdata := cannotconv
      END;
     numeral :
      BEGIN
       IF (TRUNC(token.tnum) <> minindex) AND
          (TRUNC(token.tnum) IN [1..mininfo ^.numdivisions])
        THEN BEGIN
         toindex := TRUNC(token.tnum);
         IF ptomeas ^.ordtype = ordered
          THEN IF NOT (toindex IN [minindex-1, minindex+1])
                THEN toindex := 0
        END;
       IF toindex = 0
        THEN WRITELN('Error, invalid group number entered')
      END;
     OTHERWISE WRITELN('Error, invalid command')
    END;
   UNTIL (token.ttype = nonetok) OR (toindex <> 0);
   WRITELN;
   IF argfreq <> freqtoolow
    THEN BEGIN
     datacomb := TRUE;
     IF minindex < toindex
      THEN BEGIN
       i := minindex;
       minindex := toindex;
       toindex := i
      END;
     IF controw ^.measused = ptomeas
      THEN combgroups(controw, toindex, minindex);
     IF contcolumn ^.measused = ptomeas
      THEN combgroups(contcolumn, toindex, minindex)
    END
  END
END; { proc trytocombgroups }


PROCEDURE regroupdata (
  VAR ptoitem : itempointer;
  VAR infohead : gpinfopointer );


  PROCEDURE regroup (
    VAR fromds,
        tods : numpointer;
    VAR noitems : INTEGER );

  VAR i,
      index : INTEGER;
      ptogroup : gpnodepointer;

  BEGIN
   FOR i := 1 TO noitems
    DO BEGIN
```

```
      ptogroup := infohead ^.grouphead;
      index := 1;
      WHILE NOT(TRUNC(fromds ^[i]) IN ptogroup ^.members)
       DO BEGIN
        ptogroup := ptogroup ^.nextnode;
        index := index + 1
       END;
      tods ^[i] := index
     END
   END; { proc regroup }

 BEGIN { regroupdata }
  WITH ptoitem ^
   DO BEGIN
    IF convdata = NIL
     THEN BEGIN
      NEW(convdata);
      regroup(attinfo ^.num_p, convdata, dsinfo ^.instances)
     END
     ELSE regroup(convdata, convdata, dsinfo ^.instances)
   END
 END; { proc regroupdata }


 PROCEDURE combassoccat (
   VAR ptolisthead : listheadpointer );

 VAR ptoitem : itempointer;

 BEGIN
  ptoitem := ptolisthead ^.itemhead;
  IF controw ^.numdivisions <> controw ^.measused ^.numofcat
   THEN regroupdata(ptoitem, controw);
  ptoitem := ptoitem ^.nextitem;
  IF contcolumn ^.numdivisions <> contcolumn ^.measused ^.numofcat
   THEN regroupdata(ptoitem, contcolumn)
 END; { proc combassoccat }


 PROCEDURE combloccat (
   VAR ptolisthead : listheadpointer );

 VAR ptoitem : itempointer;

 BEGIN
  ptoitem := ptolisthead ^.itemhead;
  WHILE ptoitem <> NIL
   DO BEGIN
    regroupdata(ptoitem, controw);
    ptoitem := ptoitem ^.nextitem
   END
 END; { proc combloccat }


 PROCEDURE checkchifreq (
   VAR ptolisthead : listheadpointer;
       testclass : classtype );

 VAR datacomb : BOOLEAN;

 BEGIN
  { form contingency table }
  IF testclass = association
```

```
   THEN formassoccont(ptolisthead)
   ELSE formloccont(ptolisthead);
  datacomb := FALSE;
  REPEAT
   IF efreqOK
    THEN argfreq := freqOK
    ELSE trytocombgroups(datacomb);
  UNTIL argfreq <> frequnknown;
  IF (argfreq = freqOK) AND datacomb
   THEN IF testclass = association
         THEN combassoccat(ptolisthead)
         ELSE combloccat(ptolisthead)
END; { proc checkchifreq }


PROCEDURE performsummary (
   VAR ptolisthead : listheadpointer );

{ summarise data levels of arguments
   in list headed by ptolisthead }

VAR ptoitem : itempointer;
    ptoatt : attnodepointer;

BEGIN
 ptoitem := ptolisthead ^.itemhead;
 WHILE (ptoitem <> NIL) AND (argsummary <> novalidstate)
  DO BEGIN
   ptoatt := ptoitem ^.attinfo;
   IF ptoatt ^.datalevel = ratio
    THEN CASE argsummary OF
          startstate : argsummary := allrat;
          allord     : argsummary := ordqnt;
          allqlt     : argsummary := nomqnt;
          OTHERWISE
         END
    ELSE IF ptoatt ^.datalevel = interval
     THEN CASE argsummary OF
           startstate, allrat : argsummary := intrat;
           allord             : argsummary := ordqnt;
           allqlt             : argsummary := nomqnt;
           OTHERWISE
          END
    ELSE IF ptoatt ^.datalevel = rank
     THEN CASE argsummary OF
           startstate..allqnt : argsummary := rankqnt;
           ordqnt, allord     : argsummary := rankqlt;
           nomqnt, allqlt     : argsummary := novalidstate;
           OTHERWISE
          END
    ELSE IF ptoatt ^.meas_p ^.meas_type = quantmeas
     { ordinal quantitative }
     THEN CASE argsummary OF
           startstate..intrat : argsummary := allqnt;
           allord             : argsummary := ordqnt;
           allqlt             : argsummary := nomqnt;
           OTHERWISE
          END
    ELSE IF ptoatt ^.meas_p ^.ordtype = ordered
     { ordinal qualitative }
     THEN CASE argsummary OF
           startstate     : argsummary := allord;
           allrat..allqnt : argsummary := ordqnt;
          rankqnt         : argsummary := rankqlt;
          OTHERWISE
         END
    ELSE IF ptoatt ^.meas_p ^.settype = closedset
     { nominal closed set }
     THEN CASE argsummary OF
          startstate, allord       : argsummary := allqlt;
          allrat..allqnt, ordqnt : argsummary := nomqnt;
          rankqnt, rankqlt         : argsummary := novalidstate;
          OTHERWISE
         END
    ELSE { nominal open set } argsummary := novalidstate;
   ptoitem := ptoitem ^.nextitem
  END
END; { proc performsummary }


{********** converting data for test routines **********}


PROCEDURE convertdata (
      ptoitem : itempointer;
  VAR ptoreqmeas : measpointer;
  VAR stateofdata : datastate );

{ check that arg in ptoitem is measured using reqmeas,
  if not see if the data can be converted,
  set stateofdata to cannotconv if conversion not possible }

VAR argmeas : word;
    convreq : BOOLEAN;
    ptoconv : convpointer;
    i : INTEGER;

BEGIN
 argmeas := ptoitem ^.attinfo ^.meas_p ^.measname;
 convreq := argmeas <> ptoreqmeas ^.measname;
 IF convreq
  THEN BEGIN
   convsearch(conv_root, ptoitem^.attinfo^.meas_p^.measname
              + ptoreqmeas^.measname, ptoconv);
   IF ptoconv = NIL
    THEN getconvinfo(ptoitem^.attinfo^.meas_p, ptoreqmeas, ptoconv);
   IF ptoconv = NIL
    THEN stateofdata := cannotconv
  END;
 IF (stateofdata <> cannotconv) AND (ptoitem ^.nextitem <> NIL)
  THEN convertdata(ptoitem ^.nextitem, ptoreqmeas, stateofdata);
 IF stateofdata <> cannotconv THEN
  IF convreq
   THEN BEGIN
    IF ptoitem ^.convdata = NIL
     THEN NEW(ptoitem ^.convdata);
    ptoitem ^.measinfo := ptoreqmeas;
    performconv(ptoconv, ptoitem ^. attinfo ^.num_p ^,
                ptoitem ^.convdata ^, ptoitem ^.dsinfo ^.instances)
   END
END; { proc convertdata }


PROCEDURE checksamemeas (
  VAR ptolisthead : listheadpointer;
      measspec : sortofmeas;
    VAR stateofdata : datastate );

  { see if all data measured using same suitable meas scheme,
    if not ask for scheme to use and check conversions possible,
    set stateofdata to appropriate value }

  VAR ptoitem : itempointer;
      ptomeas : measpointer;
      meastouse : word;

    PROCEDURE checkconv;

    BEGIN
     stateofdata := convOK;
     convertdata(ptolisthead ^.itemhead, ptomeas, stateofdata);
     IF stateofdata = cannotconv
      THEN stateofdata := dataunknown
    END; { proc checkconv }

  BEGIN { checksamemeas }
   ptoitem := ptolisthead ^.itemhead;
   ptomeas := ptoitem ^.attinfo ^.meas_p;
   REPEAT
    ptoitem := ptoitem ^.nextitem;
    IF ptomeas <> ptoitem ^.attinfo ^.meas_p
     THEN ptomeas := NIL;
   UNTIL (ptoitem ^.nextitem = NIL) OR (ptomeas = NIL);
   IF ptomeas <> NIL
    THEN IF meastypeOK(ptomeas, measspec)
     THEN stateofdata := origOK;
   IF stateofdata = dataunknown
    THEN BEGIN
     WRITELN;
     WRITELN('Each argument needs to be measured with the same');
     CASE measspec OF
      quant     :
        WRITELN('  quantitative measurement scheme');
      ordqual   :
        WRITELN('  ordered qualitative measurement scheme');
      unordqual :
        WRITELN('  unordered qualitative measurement scheme');
      orddich   :
        WRITELN('  ordered dichotomous measurement scheme');
      unorddich :
        WRITELN('  unordered dichotomous measurement scheme')
     END;
     REPEAT
      READLN;
      WRITE('Enter measurement scheme to use (or NONE) : ');
      gettoken;
      WRITELN;
      CASE token.ttype OF
       nonetok : stateofdata := cannotconv;
       showargmeas  : displayargmeas(ptolisthead);
       showcandmeas : displaycandmeas(measspec);
       identifier :
        BEGIN
         meastouse := token.tchars;
         meassearch(meas_root, meastouse, ptomeas);
         IF ptomeas = NIL
          THEN BEGIN
           IF wantstodecmeas
            THEN BEGIN
          IF measspec = quant
           THEN getquantinfo(meastouse, ptomeas)
           ELSE getqualinfo(meastouse, measspec, ptomeas);
          checkconv
         END
       END
       ELSE IF ptomeas <> NIL
        THEN IF meastypeOK(ptomeas, measspec)
         THEN checkconv
         ELSE WRITELN('Error, measurement scheme entered is',
                      ' not of an appropriate type')
     END;
    OTHERWISE WRITELN('Error, invalid command')
   END;
  UNTIL stateofdata <> dataunknown
 END
END; { proc checksamemeas }


PROCEDURE disposegenqlt (
  VAR ptolisthead : listheadpointer );

VAR ptoitem : itempointer;

BEGIN
 ptoitem := ptolisthead ^.itemhead;
 WHILE ptoitem <> NIL
  DO WITH ptoitem ^ DO BEGIN
   IF measinfo <> NIL
    THEN IF measinfo ^.meas_type = qualmeas
          THEN BEGIN
           DISPOSE(convdata);
           convdata := NIL;
           measinfo := NIL
          END;
   ptoitem := nextitem
  END;
 qltdata := dataunknown
END; { proc disposegenqlt }


PROCEDURE categoriseqnt (
  VAR ptoitem : itempointer;
  VAR stateofdata : datastate );

{ for nomcat, work through args any for any which are quant
  find out if the data can be categorised and how }

VAR ptoargmeas,
    ptoreqmeas : measpointer;
    ptoconv : convpointer;
    meastouse : word;

  PROCEDURE checkconv;

  BEGIN
   convsearch(conv_root, ptoargmeas ^.measname +
              ptoreqmeas ^.measname, ptoconv);
   IF ptoconv = NIL
    THEN getconvinfo(ptoargmeas, ptoreqmeas, ptoconv);
   IF ptoconv = NIL
    THEN stateofdata := dataunknown
    ELSE stateofdata := convOK
   END; { proc checkconv }

 BEGIN { categoriseqnt }
  ptoargmeas := ptoitem ^.attinfo ^.meas_p;
  ptoconv := NIL;
  IF ptoargmeas ^.meas_type = quantmeas
   THEN BEGIN
    WRITELN;
    WRITE('Attribute ');
    displayarg(ptoitem);
    WRITELN(' is measured in ', ptoargmeas ^.measname);
    WRITELN('and needs to be converted to an ordered',
             ' qualitative scheme');
    REPEAT
     READLN;
     WRITE('Enter measurement scheme to use (or NONE) : ');
     gettoken;
     CASE token.ttype OF
      nonetok : stateofdata := cannotconv;
      showcandmeas : displaycandmeas(ordqual);
      identifier :
       BEGIN
        meastouse := token.tchars;
        meassearch(meas_root, meastouse, ptoreqmeas);
        IF ptoreqmeas = NIL
         THEN BEGIN
          IF wantstodecmeas
           THEN BEGIN
            getqualinfo(meastouse, ordqual, ptoreqmeas);
            checkconv
           END
         END
         ELSE IF ptoreqmeas <> NIL
          THEN IF meastypeOK(ptoreqmeas, ordqual)
           THEN checkconv
           ELSE WRITELN('Error, measurement scheme entered',
                        ' is not of an appropriate type')
       END;
      OTHERWISE WRITELN('Error, invalid command')
     END;
    UNTIL stateofdata <> dataunknown;
    WRITELN
   END;
  IF (stateofdata <> cannotconv) AND (ptoitem ^.nextitem <> NIL)
   THEN categoriseqnt(ptoitem ^.nextitem, stateofdata);
  IF (ptoconv <> NIL) AND (stateofdata <> cannotconv)
   THEN BEGIN
    NEW(ptoitem ^.convdata);
    ptoitem ^.measinfo := ptoreqmeas;
    performconv(ptoconv, ptoitem ^.attinfo ^.num_p ^,
                ptoitem ^.convdata ^, ptoitem ^.dsinfo ^.instances)
   END
 END; { proc categoriseqnt }


PROCEDURE dichqltdata (
  VAR ptolisthead : listheadpointer );

VAR ptomeas : measpointer;

BEGIN
 WITH ptolisthead ^.itemhead ^
  DO IF measinfo <> NIL
```

```
      THEN ptomeas := measinfo
      ELSE ptomeas := attinfo ^.meas_p;
 IF meastypeOK(ptomeas, dich)
  THEN dichdata := qltdata
  ELSE BEGIN
   combloccat(ptolisthead);
   dichdata := convOK
  END
END; { proc dichqltdata }


{********** test level routines **********}


PROCEDURE validatetest (
  VAR candtest : valid_tests;
      ptocheck : checkpointer;
  VAR ptolisthead : listheadpointer;
  VAR ptofailcheck : checkpointer );

{ apply checks headed by ptocheck to arguments in list
  ptolisthead and set ptofailcheck to any checknode
  where a requirement cannot be met }

VAR testOK : BOOLEAN;

BEGIN
 testOK := TRUE;
 WHILE (ptocheck <> NIL) AND testOK
  DO BEGIN
   CASE ptocheck ^.semcheck OF
    twosample :
      checknumargs(twosample, ptolisthead ^.no_items, testOK);
    relatedinst :
     BEGIN
      IF argsrel = relunknown
       THEN checkrelargs(ptolisthead);
      testOK := argsrel = related
     END;
    normal :
     BEGIN
      IF argsnormal = normunknown
       THEN checknormalargs(ptolisthead);
      testOK := argsnormal IN [normalOK, assnormal]
     END;
    eqvar :
     BEGIN
      IF argvar = varunknown
       THEN checkeqvar(ptolisthead);
      testOK := argvar IN [eqvarOK, asseqvar]
     END;
    nige30 :
     BEGIN
      IF numinst = instunknown
       THEN checknige30(ptolisthead);
      testOK := numinst = instOK
     END;
    chifreq :
     BEGIN
      IF argfreq = frequnknown
       THEN BEGIN
        IF candtest IN [pearson..coeff_of_cont]
         THEN checkchifreq(ptolisthead, association)
     ELSE checkchifreq(ptolisthead, location)
    END;
    testOK := argfreq = freqOK
  END;
 eqratqnt..eqdichcat :
  BEGIN
   IF argsummary = startstate
    THEN performsummary(ptolisthead);
   CASE ptocheck ^.semcheck OF
    eqratqnt :
      IF argsummary = allrat
       THEN BEGIN
        IF qntdata = dataunknown
         THEN checksamemeas(ptolisthead, quant, qntdata);
        IF qntdata = cannotconv
         THEN testOK := FALSE
         ELSE IF qltdata = convOK
               THEN disposegenqlt(ptolisthead)
       END
       ELSE testOK := FALSE;
    ratqnt :
      IF argsummary <> allrat
       THEN testOK := FALSE
       ELSE IF qltdata = convOK
             THEN disposegenqlt(ptolisthead);
    eqintqnt :
      IF argsummary IN [allrat, intrat]
       THEN BEGIN
        IF qntdata = dataunknown
         THEN checksamemeas(ptolisthead, quant, qntdata);
        IF qntdata = cannotconv
         THEN testOK := FALSE
         ELSE IF qltdata = convOK
               THEN disposegenqlt(ptolisthead)
       END
       ELSE testOK := FALSE;
    intqnt :
      IF NOT (argsummary IN [allrat, intrat])
       THEN testOK := FALSE
       ELSE IF qltdata = convOK
             THEN disposegenqlt(ptolisthead);
    ranked :
      IF NOT (argsummary IN [allrat..rankqnt])
       THEN testOK := FALSE;
    eqordqnt :
     IF argsummary IN [allrat..allqnt]
      THEN BEGIN
       IF qntdata = dataunknown
        THEN checksamemeas(ptolisthead, quant, qntdata);
       IF qntdata = cannotconv
        THEN testOK := FALSE
        ELSE IF qltdata = convOK
              THEN disposegenqlt(ptolisthead)
      END
      ELSE testOK := FALSE;
    eqordqlt :
     IF argsummary IN [allrat..allqnt, ordqnt, allord]
      THEN BEGIN
       IF (argsummary IN [allrat..allqnt])
          AND (qntdata = dataunknown)
        THEN BEGIN
         checksamemeas(ptolisthead, quant, qntdata);
         IF (qntdata <> cannotconv) AND (qltdata = convOK)
        THEN disposegenqlt(ptolisthead)
      END;
     IF NOT (qntdata IN [origOK, convOK])
            AND (qltdata = dataunknown)
      THEN BEGIN
       checksamemeas(ptolisthead, ordqual, qltdata);
       IF qltdata = cannotconv THEN dichdata := cannotconv
      END;
     testOK := (qntdata IN [origOK, convOK]) OR
               (qltdata IN [origOK, convOK])
   END
   ELSE testOK := FALSE;
 ordqlt :
   IF NOT (argsummary IN [allrat..allord])
    THEN testOK := FALSE
    ELSE IF qltdata = convOK
          THEN disposegenqlt(ptolisthead);
 eqnomcat :
  BEGIN
   IF (argsummary IN [allrat..allqnt, ordqnt..allord])
       AND (qltdata = dataunknown)
    THEN BEGIN
     checksamemeas(ptolisthead, ordqual, qltdata);
     IF qltdata = cannotconv THEN dichdata := cannotconv
    END
   ELSE IF (argsummary = allqlt) AND (qltdata = dataunknown)
    THEN BEGIN
     checksamemeas(ptolisthead, unordqual, qltdata);
     IF qltdata = cannotconv THEN dichdata := cannotconv
    END;
   testOK := qltdata IN [origOK, convOK]
  END;
 nomcat :
   IF (argsummary IN [rankqnt, rankqlt, novalidstate])
      OR (qltdata = cannotconv)
    THEN testOK := FALSE
    ELSE IF (argsummary IN [allrat..allqnt, ordqnt, nomqnt])
             AND (qltdata = dataunknown)
     THEN BEGIN
      categoriseqnt(ptolisthead ^.itemhead, qltdata);
      IF qltdata = cannotconv
       THEN BEGIN
        testOK := FALSE;
        dichdata := cannotconv
       END
     END;
 eqdichcat :
  BEGIN
   IF NOT (argsummary IN [rankqnt, rankqlt, novalidstate])
     AND (dichdata = dataunknown)
    THEN BEGIN
     IF qltdata IN [origOK, convOK]
      THEN dichqltdata(ptolisthead)
      ELSE IF (argsummary IN [allrat..allqnt, ordqnt..allord])
       THEN BEGIN
        checksamemeas(ptolisthead, orddich, dichdata);
        IF dichdata = cannotconv
         THEN qltdata := cannotconv
       END
       ELSE checksamemeas(ptolisthead, unorddich, dichdata)
    END;
   testOK := dichdata IN [origOK, convOK]
  END
```

```
        END
       END
     END;
     IF testOK THEN ptocheck := ptocheck ^.nextcheck
    END;
   IF testOK
    THEN ptofailcheck := NIL
    ELSE ptofailcheck := ptocheck
  END; { proc validatetest }


  PROCEDURE showtestreq (
    VAR testtoapply : valid_tests;
        ptocheck : checkpointer );

  BEGIN
   WRITELN;
   WRITELN('The requirements for ',testtoapply:16,
           ' are as follows :-');
   WRITELN;
   WHILE ptocheck <> NIL
    DO WITH ptocheck ^ DO BEGIN
     CASE semcheck OF
      twosample :
       WRITELN('  The test is used in a two sample situation');
      relatedinst :
       WRITELN('  The instances of each sample should be related');
      normal :
       WRITELN('  The data in each sample should be normally',
               ' distributed');
      eqvar :
       WRITELN('  The variance of each sample should be equal');
      nige30 :
       WRITELN('  Each sample should be measured for at least 30',
               ' instances');
      chifreq :
       BEGIN
        WRITELN('  The frequencies for each of the measurement');
        WRITELN('    scheme categories should not be too small')
       END;
      eqdichcat :
       WRITELN('  The data should be formed into a dichotomy',
               ' in the same way');
      nomcat :
       BEGIN
        WRITELN('  The data should be measured using closed');
        WRITELN('    qualitative measurement schemes')
       END;
      eqnomcat :
       BEGIN
        WRITELN('  The data should be measured using the same');
        WRITELN('    closed qualitative measurement scheme')
       END;
      ordqlt :
       WRITELN('  The data should be measured on at least an',
               ' ordinal scale');
      eqordqlt :
       BEGIN
        WRITELN('  The data should be measured on at least an');
        WRITELN('    ordinal scale using the same measurement scheme')
       END;
      ranked :
       WRITELN('  The data should be quantitative or rank');
    eqordqnt :
     BEGIN
      WRITELN('  The data should be quantitative and measured');
      WRITELN('    using the same measurement scheme')
     END;
    intqnt :
     WRITELN('  The data should be measured on at least an',
             ' interval scale');
    eqintqnt :
     BEGIN
      WRITELN('  The data should be measured on at least an');
      WRITELN('    interval scale using the same measurement',
              ' scheme')
     END;
    ratqnt :
     WRITELN('  The data should be measured on a ratio scale');
    eqratqnt :
     BEGIN
      WRITELN('  The data should be measured on a ratio scale');
      WRITELN('    using the same measurement scheme')
     END
   END;
   ptocheck := nextcheck
  END
END; { proc showtestreq }


PROCEDURE expfailedcheck (
  VAR semcheck : validreqs );

BEGIN
 WRITELN;
 WRITE('The requested test was not acceptable because ');
 CASE semcheck OF
  twosample :
   BEGIN
    WRITELN('more than two');
    WRITELN('  arguments have been given')
   END;
  relatedinst :
   BEGIN
    WRITELN('the arguments');
    WRITELN('  are not related')
   END;
  normal :
   BEGIN
    WRITELN('normality');
    WRITELN('  cannot be assumed for each sample')
   END;
  eqvar :
   BEGIN
    WRITELN('equality of');
    WRITELN('  variances cannot be assumed')
   END;
  nige30 :
   BEGIN
    WRITELN('each argument');
    WRITELN('  does not have 30 or more instances')
   END;
  chifreq :
   BEGIN
    WRITELN('the frequencies');
    WRITELN('  of the measurement scheme categories are too small')
    END;
   eqdichcat :
    IF dichdata = cannotconv
     THEN BEGIN
      WRITELN;
      WRITELN('  the data cannot be converted into a suitable form')
     END
     ELSE BEGIN
      WRITELN;
      WRITELN('  the level of measurement of the data is unsuitable')
     END;
   nomcat, eqnomcat :
    IF qltdata = cannotconv
     THEN BEGIN
      WRITELN;
      WRITELN('  the quantitative data cannot be categorised')
     END
     ELSE BEGIN
      WRITELN;
      WRITELN('  the level of measurement of the data is unsuitable')
     END;
   ordqlt, ranked, intqnt, ratqnt :
    BEGIN
     WRITELN;
     WRITELN('  the level of measurement of the data is unsuitable')
    END;
   eqordqlt :
    IF qltdata = cannotconv
     THEN BEGIN
      WRITELN;
      WRITELN('  the data cannot be converted into a suitable form')
     END
     ELSE BEGIN
      WRITELN;
      WRITELN('  the level of measurement of the data is unsuitable')
     END;
   eqordqnt, eqintqnt, eqratqnt :
    IF qntdata = cannotconv
     THEN BEGIN
      WRITELN;
      WRITELN('  the data cannot be converted into a suitable form')
     END
     ELSE BEGIN
      WRITELN;
      WRITELN('  the level of measurement of the data is unsuitable')
     END
 END
END; { proc expfailedcheck }


PROCEDURE displaycombcats;

  PROCEDURE showgroups (
    VAR infohead : gpinfopointer );

  VAR i, j, numonline : INTEGER;
      ptogroup : gpnodepointer;

  BEGIN
   WITH infohead ^
    DO BEGIN
     WRITELN('Categories in ',
             measused ^.measname:strlen(measused ^.measname),
                  ' have been grouped as follows :-');
       i := 1;
       ptogroup := grouphead;
       WHILE ptogroup <> NIL
        DO WITH ptogroup ^ DO BEGIN
         WRITE('   (');
         numonline := 0;
         FOR j := 1 TO measused ^.numofcat
          DO IF j IN members
           THEN BEGIN
            IF numonline = 3
             THEN BEGIN
              WRITELN;
              WRITE('    ');
              numonline := 0
             END;
            CASE measused ^.cattype OF
             identifier : WRITE(charcatvalue(measused, j):18);
             numeral    : WRITE(numcatvalue(measused, j):9:2)
            END;
            numonline := numonline + 1
           END;
         WRITELN(')');
         i := i + 1;
         ptogroup := nextnode
        END
      END;
     WRITELN
    END; { proc showgroups }

  BEGIN { displaycombcats }
   IF controw ^.numdivisions <> controw ^.measused ^.numofcat
    THEN showgroups(controw);
   IF contcolumn ^.measused <> NIL
    THEN IF (contcolumn ^.measused <> controw ^.measused) AND
            (contcolumn ^.numdivisions <>
             contcolumn ^.measused ^.numofcat)
          THEN showgroups(contcolumn)
  END; { proc displaycombcats }


  PROCEDURE displayconvargs (
    VAR ptolisthead : listheadpointer );

  VAR ptoitem : itempointer;

  BEGIN
   ptoitem := ptolisthead ^.itemhead;
   WHILE ptoitem <> NIL
    DO WITH ptoitem ^ DO BEGIN
     IF measinfo <> NIL
      THEN BEGIN
       displayarg(ptoitem);
       WRITELN(' converted from ', attinfo ^.meas_p ^.measname
                :strlen(attinfo ^.meas_p ^.measname),
                ' to ', measinfo ^.measname)
      END;
     ptoitem := nextitem
    END;
   WRITELN
  END; { proc displayconvargs }
```

```
PROCEDURE reviewtestchecks (
  VAR ptolisthead : listheadpointer;
      ptocheck : checkpointer );

BEGIN
 WHILE ptocheck <> NIL
  DO WITH ptocheck ^ DO BEGIN
   CASE semcheck OF
    normal :
      IF argsnormal = assnormal
       THEN BEGIN
        WRITELN('Warning, care should be taken when interpreting');
        WRITELN('  the results since the test assumes that the');
        WRITELN('  data is normally distributed');
        WRITELN
       END;
    eqvar :
      IF argvar = asseqvar
       THEN BEGIN
        WRITELN('Warning, care should be taken when interpreting');
        WRITELN('  the results since the test assumes that the');
        WRITELN('  sample variances are equal');
        WRITELN
       END;
    chifreq : displaycombcats;
    eqratqnt, eqintqnt, eqordqnt, eqordqlt :
      IF explain AND ((qntdata = convOK) OR (qltdata = convOK))
       THEN BEGIN
        WRITELN('The data has been converted to the same ',
                'measurement scheme');
        displayconvargs(ptolisthead)
       END;
    eqnomcat :
      IF explain AND (qltdata = convOK)
       THEN BEGIN
        WRITELN('The data has been converted to the same ',
                'qualitative measurement scheme');
        displayconvargs(ptolisthead)
       END;
    nomcat :
      IF explain AND (qltdata = convOK)
       THEN BEGIN
        WRITELN('All quantitative data has been converted ',
                'to qualitative data');
        displayconvargs(ptolisthead)
       END;
    eqdichcat :
      IF explain AND (dichdata = convOK)
       THEN BEGIN
        IF qltdata = convOK
         THEN BEGIN
          WRITELN('The data has been converted to the same ',
                  'qualitative measurement scheme');
          displayconvargs(ptolisthead);
          displaycombcats
         END
        ELSE IF qltdata = origOK
         THEN displaycombcats
        ELSE BEGIN
         WRITELN('The data had been converted to the same ',
                 'dichotomous measurement scheme');
         displayconvargs(ptolisthead)
        END
        END
      END;
      ptocheck := nextcheck
     END
  END; { proc reviewtestchecks }


  PROCEDURE disposecontinfo (
    VAR infohead : gpinfopointer );

  VAR ptogroup : gpnodepointer;

  BEGIN
   WHILE infohead ^.grouphead <> NIL
    DO BEGIN
     ptogroup := infohead ^.grouphead;
     infohead ^.grouphead := ptogroup ^.nextnode;
     DISPOSE(ptogroup)
    END;
   DISPOSE(infohead)
  END; { disposecontinfo }


  PROCEDURE checktestreq (
    VAR testclass : classtype;
    VAR testchecks : ARRAY[firsttest..lasttest:statcomms]
                       OF checkpointer;
    VAR usertest : valid_tests );

  VAR testclassstr : textmessage;
      ptolisthead : listheadpointer;
      testtoapply : valid_tests;
      state : (searching, testfound, searchfailed);
      showargs : BOOLEAN;
      ptofailcheck : checkpointer;

  BEGIN
   CASE testclass OF
    association : testclassstr := 'measure of association ';
    location    : testclassstr := 'test of location '
   END;
   showargs := listheadhead ^.nexthead <> NIL;
   ptolisthead := listheadhead;
   WHILE ptolisthead <> NIL
    DO BEGIN
     IF showargs
      THEN BEGIN
       WRITELN;
       WRITELN('Considering the following arguments :-');
       WRITELN;
       displayarglist(ptolisthead)
      END;
     { initialise variables for current list }
     testtoapply := usertest;
     argsummary := startstate;
     argsrel := relunknown;
     argsnormal := normunknown;
     argvar := varunknown;
     numinst := instunknown;
     argfreq := frequnknown;
     controw := NIL;
     contcolumn := NIL;
     conttotal := 0;
```

```
qntdata := dataunknown;
qltdata := dataunknown;
dichdata := dataunknown;
state := searching;
IF testtoapply <> nulltest
 THEN BEGIN
  { see if user specified test can be applied }
  validatetest(testtoapply, testchecks[testtoapply],
               ptolisthead, ptofailcheck);
  IF ptofailcheck = NIL
   THEN state := testfound
   ELSE BEGIN
    IF explain
     THEN BEGIN
      showtestreq(testtoapply, testchecks[testtoapply]);
      expfailedcheck(ptofailcheck ^.semcheck)
     END
     ELSE BEGIN
      WRITELN;
      WRITELN('User specified test cannot be applied')
     END;
    WRITELN;
    REPEAT
     READLN;
     WRITE('Do you wish to search for a ',
           testclassstr, '(yes/no) : ');
     gettoken;
    UNTIL token.ttype IN [yestok, notok];
    WRITELN;
    IF token.ttype = notok
     THEN state := searchfailed
   END
 END;
IF state = searching THEN testtoapply := firsttest;
WHILE state = searching { consider next test }
 DO BEGIN
  validatetest(testtoapply, testchecks[testtoapply],
               ptolisthead, ptofailcheck);
  IF ptofailcheck = NIL
   THEN BEGIN
    WRITELN;
    WRITELN('Recommended ',testclassstr,'is ',testtoapply:16);
    IF explain
     THEN showtestreq(testtoapply, testchecks[testtoapply]);
    WRITELN;
    REPEAT
     READLN;
     WRITE('Do you wish to apply this test (yes/no) : ');
     gettoken;
    UNTIL token.ttype IN [yestok, notok];
    WRITELN;
    IF token.ttype = yestok
     THEN state := testfound
     ELSE state := searchfailed
   END
   ELSE IF testtoapply = lasttest
    THEN BEGIN
     state := searchfailed;
     WRITELN;
     WRITELN('End of list reached, cannot get data into a form');
     WRITELN('to apply a ',testclassstr)
    END
    ELSE testtoapply := SUCC(testtoapply)
```

```
     END;
    IF state = testfound
     THEN BEGIN
      WRITELN;
      reviewtestchecks(ptolisthead, testchecks[testtoapply]);
      WRITELN('Call proc to apply ',testtoapply)
     END;
    IF conttotal <> 0
     THEN BEGIN
      disposecontinfo(controw);
      disposecontinfo(contcolumn)
     END;
   ptolisthead := ptolisthead ^.nexthead
  END
END; { proc checktestreq }


{********** type of test level routines **********}


PROCEDURE removeitem (
  VAR itemtoremove : itempointer;
  VAR ptolisthead : listheadpointer );

{ removes itemtoremove from the list
  of items in ptolisthead }

VAR item : itempointer;

BEGIN
 item := ptolisthead ^.itemhead;
 WHILE item ^.nextitem <> itemtoremove
  DO item := item ^.nextitem;
 item ^.nextitem := itemtoremove ^.nextitem;
 itemtoremove ^.nextitem := NIL;
 ptolisthead ^.no_items := ptolisthead ^.no_items - 1
END; { proc removeitem }


PROCEDURE createlist (
  VAR itemtomove : itempointer;
  VAR currentlisthead,
      lastlisthead : listheadpointer );

{ creates a new list head after lastlisthead
  and moves itemtomove from currentlisthead
  into the new list }

VAR newlisthead : listheadpointer;

BEGIN
 removeitem(itemtomove, currentlisthead);
 NEW(newlisthead);
 WITH newlisthead ^
  DO BEGIN
   nexthead := lastlisthead ^.nexthead;
   no_items := 1;
   itemhead := itemtomove
  END;
 lastlisthead ^.nexthead := newlisthead;
 lastlisthead := newlisthead
END; { proc createlist }
```

```
PROCEDURE addtolist (
  VAR itemtomove : itempointer;
  VAR oldlisthead,
      newlisthead : listheadpointer );

{ moves itemtomove from oldlisthead to newlisthead }

VAR item : itempointer;

BEGIN
 removeitem(itemtomove, oldlisthead);
 item := newlisthead ^.itemhead;
 WHILE item ^.nextitem <> NIL
  DO item := item ^.nextitem;
 item ^.nextitem := itemtomove;
 newlisthead ^.no_items := newlisthead ^.no_items + 1
END; { proc addtolist }


PROCEDURE disposeoflist (
  VAR headoflists : listheadpointer );

{ dispose of listhead and item nodes in headoflists }

VAR ptolisthead : listheadpointer;
    item : itempointer;

BEGIN
 WHILE headoflists <> NIL
  DO BEGIN
   ptolisthead := headoflists;
   headoflists := ptolisthead ^.nexthead;
   WHILE ptolisthead ^.itemhead <> NIL
    DO BEGIN
     item := ptolisthead ^.itemhead;
     ptolisthead ^.itemhead := item ^.nextitem;
     IF item ^.convdata <> NIL
      THEN DISPOSE(item ^.convdata);
     DISPOSE(item)
    END;
   DISPOSE(ptolisthead)
  END
END; { proc disposeoflist }


PROCEDURE checkclassreq (
  currentcheck : checkpointer );

{ perform checks for class of tests on given
  arguments in listheadhead }

VAR listtocheck,
    nextlisthead,
    lastlisthead,
    reqlisthead : listheadpointer;
    itemtocheck,
    nextlistitem : itempointer;
    state : (listnotfound, samelist, difflist, newlist);
    checkOK : BOOLEAN;

BEGIN
 WHILE currentcheck <> NIL
```

```
   DO BEGIN
    listtocheck := listheadhead;
    WHILE listtocheck <> NIL
     DO BEGIN
      nextlisthead := listtocheck ^.nexthead;
      lastlisthead := listtocheck;
      itemtocheck := listtocheck ^.itemhead ^.nextitem;
      WHILE itemtocheck <> NIL
       DO BEGIN
        reqlisthead := listtocheck;
        state := listnotfound;
        nextlistitem := itemtocheck ^.nextitem;
        WHILE state = listnotfound
         DO BEGIN
          CASE currentcheck ^.semcheck OF
           eqdomains   :
            checkeqdom(reqlisthead ^.itemhead, itemtocheck, checkOK);
           relatedinst :
            checkrelinsts(reqlisthead ^.itemhead ^.dsinfo,
                          itemtocheck ^.dsinfo, checkOK);
           simenttype  :
            checkenttype(reqlisthead ^.itemhead,
                         itemtocheck, checkOK)
          END;
          IF checkOK
           THEN BEGIN
            IF reqlisthead = listtocheck
             THEN state := samelist
             ELSE state := difflist
           END
           ELSE IF reqlisthead = lastlisthead
            THEN state := newlist
            ELSE reqlisthead := reqlisthead ^.nexthead
         END;
        IF state = newlist
         THEN createlist(itemtocheck, listtocheck, lastlisthead)
        ELSE IF state = difflist
         THEN addtolist(itemtocheck, listtocheck, reqlisthead);
        itemtocheck := nextlistitem
       END;
      listtocheck := nextlisthead
     END;
    currentcheck := currentcheck ^.nextcheck
   END
END; { proc checkclassreq }


PROCEDURE expclassreqs (
  VAR testclassstr : textmessage;
      ptocheck : checkpointer );

BEGIN
 WRITELN;
 WRITELN('The requirement(s) for a ', testclassstr,
         'are as follows :-');
 WRITELN;
 WHILE ptocheck <> NIL
  DO WITH ptocheck ^ DO BEGIN
   CASE semcheck OF
    eqdomains :
     WRITELN('  Each sample should be measuring the ',
             'same quality or quantity.');
    relatedinst :
     WRITELN('  The instances of each sample should be related.');
    simenttype :
     WRITELN('  Each sample should be measured for ',
             'the same type of entity.')
   END;
   ptocheck := nextcheck
  END
END; { proc expclassreqs }


PROCEDURE reviewclasschecks (
  VAR testclass : classtype;
  VAR typeoftestargs : testtype );

{ explain result of performing class checks }

VAR testclassstr : textmessage;
    invalidlists,
    lastinvalid,
    validlists,
    lastvalid,
    ptolisthead,
    nextlisthead : listheadpointer;
    numargsOK : BOOLEAN;

  PROCEDURE addtolist (
    VAR head,
        last,
        current : listheadpointer );

  BEGIN
   IF head = NIL
    THEN head := current
    ELSE last ^.nexthead := current;
   last := current;
   last ^.nexthead := NIL
  END; { proc addtolist }

BEGIN { reviewclasschecks }
 CASE testclass OF
  association : testclassstr := 'measure of association ';
  location    : testclassstr := 'test of location '
 END;
 invalidlists := NIL;
 validlists := NIL;
 ptolisthead := listheadhead;
 WHILE ptolisthead <> NIL
  DO BEGIN
   nextlisthead := ptolisthead ^.nexthead;
   checknumargs(typeoftestargs, ptolisthead ^.no_items, numargsOK);
   IF numargsOK
    THEN addtolist(validlists, lastvalid, ptolisthead)
    ELSE addtolist(invalidlists, lastinvalid, ptolisthead);
   ptolisthead := nextlisthead
  END;
 IF validlists = NIL
  THEN BEGIN
   IF explain
    THEN expclassreqs(testclassstr, class_checks[testclass]);
   WRITELN;
   WRITELN('No ', testclassstr, 'can be applied.')
  END
 ELSE IF (validlists ^.nexthead <> NIL) OR (invalidlists <> NIL)
```

```
   THEN BEGIN
    { more than one valid list or an invalid list,
      need to explain situation to user }
    IF explain
     THEN expclassreqs(testclassstr, class_checks[testclass]);
    IF invalidlists <> NIL
     THEN BEGIN
      WRITELN;
      WRITELN('Cannot apply a ',testclassstr,
              'to the following argument(s)');
      WRITELN;
      ptolisthead := invalidlists;
      WHILE ptolisthead <> NIL
       DO BEGIN
        displayarglist(ptolisthead);
        ptolisthead := ptolisthead ^.nexthead
       END
     END;
    WRITELN;
    WRITE('Can apply a ', testclassstr);
    IF validlists ^.nexthead = NIL
     THEN WRITELN('to the remaining arguments')
     ELSE WRITELN('to each of the following groups');
    WRITELN;
    ptolisthead := validlists;
    WHILE ptolisthead <> NIL
     DO BEGIN
      displayarglist(ptolisthead);
      WRITELN;
      ptolisthead := ptolisthead ^.nexthead
     END;
    REPEAT
     READLN;
     WRITE('Do you wish to continue (yes/no) : ');
     gettoken;
    UNTIL token.ttype IN [yestok, notok];
    WRITELN;
    IF token.ttype = notok
     THEN disposeoflist(validlists)
   END;
 listheadhead := validlists;
 disposeoflist(invalidlists)
 { will apply a testclass to each list in listheadhead
   which will be NIL if no tests are to be applied }
END; { proc reviewclasschecks }


{********** preliminary and controlling routines **********}


PROCEDURE genitemnode (
  VAR head,
      current : itempointer;
  VAR ptods : dsnodepointer;
  VAR ptoatt : attnodepointer );

{ generate itemnode for an argument of a test, add to
  list headed by head and set current to point to it }

BEGIN
 IF head = NIL
  THEN BEGIN
   NEW(head);
```

```
     current := head
    END
    ELSE BEGIN
     NEW(current ^.nextitem);
     current := current ^.nextitem
    END;
   WITH current ^
    DO BEGIN
     dsinfo := ptods;
     attinfo := ptoatt;
     convdata := NIL;
     measinfo := NIL;
     nextitem := NIL
    END
  END; { proc genitemnode }


  PROCEDURE procstatreq (
    testclass : classtype;
    testname : valid_tests;
    typeoftestargs : testtype );

  { process a user request to perform a statistical test:
    read in arguments; perform class checks; review result
    of class checks; perform any tests }

  LABEL endofproc;
  VAR   dsname,
        attname : word;
        ptods : dsnodepointer;
        ptoatt : attnodepointer;
        ptoitem : itempointer;
        numargsOK : BOOLEAN;

    PROCEDURE dealwitherror (
      errorstate : errortype;
      errorarg : textmessage );

    BEGIN
     reporterror(errorstate, errorarg);
     GOTO endofproc
    END; { proc dealwitherror }


  BEGIN { procstatreq }
   { initialise listheadhead }
   NEW(listheadhead);
   WITH listheadhead ^
    DO BEGIN
     nexthead := NIL;
     no_items := 0;
     itemhead := NIL;
     { read in arguments }
     gettoken;
     WHILE token.ttype <> endofline
      DO BEGIN
       IF token.ttype <> identifier
        THEN dealwitherror(invarg, nullname);
       dsname := token.tchars;
       gettoken;
       IF token.ttype <> dot
        THEN dealwitherror(invarg, nullname);
       gettoken;
      IF token.ttype <> identifier
       THEN dealwitherror(invarg, nullname);
      attname := token.tchars;
      dstypesearch(ds_root, dsname, ptods);
      IF ptods = NIL
       THEN dealwitherror(dsmiss, dsname);
      IF ptods ^.attchain = NIL
       THEN dealwitherror(noatts, dsname);
      atttypesearch(ptods ^.attchain, attname, ptoatt);
      IF ptoatt = NIL
       THEN dealwitherror(attmiss, substr(dsname,1,strlen(dsname))
                          + '.' + attname);
      IF ptods ^.instances < 3
       THEN dealwitherror(insuffinst, dsname);
      genitemnode(itemhead, ptoitem, ptods, ptoatt);
      no_items := no_items + 1;
      gettoken
     END;
    checknumargs(typeoftestargs, no_items, numargsOK);
    IF NOT numargsOK
     THEN IF typeoftestargs = twosample
           THEN dealwitherror(invnumarg, '2')
           ELSE dealwitherror(invnumarg, '2+')
  END;
 WRITELN;
 checkclassreq(class_checks[testclass]);
 reviewclasschecks(testclass, typeoftestargs);
 IF listheadhead <> NIL
  THEN CASE testclass OF
        association : checktestreq(testclass, assoc_checks,
                                   testname);
        location    : checktestreq(testclass, loc_checks, testname)
       END;
 endofproc: ;
 disposeoflist(listheadhead)
END; { proc procstatreq }


{ file check_routines.pas }
```

B.4 Keyworddir.dat

```
ADDATT            addatt
ADDDS             addds
ADDENT            addent
ADDINST           addinst
ADDMEAS           addmeas
SHOWARGMEAS       showargmeas
SHOWATT           showatt
SHOWCANDMEAS      showcandmeas
SHOWDSDIR         showdsdir
SHOWENTDIR        showentdir
SHOWINST          showinst
SHOWMEAS          showmeas
SHOWMEASDIR       showmeasdir
EXPLAIN           exptok
NOEXPLAIN         noexptok
QUIT              quit
ASSOCIATION       association
LOCATION          location
PEARSON           pearson
SPEARMAN          spearman
KENDALL           kendall
TAU_C             tau_c
CRAMERS_V         cramers_v
COEFF_OF_CONT     coeff_of_cont
NORMAL_TEST       normal_test
T_PAIRED          t_paired
RANDOMISED_BLOCK  randomised_block
T_COMMON          t_common
T_SEPARATE        t_separate
ONE_WAY_AOV       one_way_aov
WILCOXON          wilcoxon
FRIEDMAN_AOV      friedman_aov
MANN_WHITNEY      mann_whitney
KRUSKAL_WALLIS    kruskal_wallis
SIGN_TEST         sign_test
MCNEMAR_TEST      mcnemar_test
COCHRAN_Q         cochran_q
CHI_SQUARED       chi_squared
FISHER_EXACT      fisher_exact
GENERIC           gentok
NONGENERIC        nongentok
OPEN              opentok
CLOSED            closedtok
YES               yestok
NO                notok
DEFAULT           deftok
TYPE              typetok
LEVEL             leveltok
MEAS              meastok
NORMAL            normtok
QUAL              qualmeas
QUANT             quantmeas
NOMINAL           nomtok
ORDINAL           ordtok
RANK              ranktok
INTERVAL          inttok
RATIO             rattok
CHARACTER         chartok
NUMERIC           numtok
```

```
MIN              min
MAX              max
UPPER            upper
NONE             nonetok
```

B.5 Classcheckdir.dat

```
association  relatedinst
location     simenttype   eqdomains
```

B.6 Assoccheckdir.dat

```
pearson         intqnt  normal
spearman        ranked
kendall         ranked
tau_c           ordqlt
cramers_v       nomcat          chifreq
coeff_of_cont   nomcat          chifreq
```

B.7 Loccheckdir.dat

```
normal_test       twosample               eqintqnt          nige30
t_paired          twosample  relatedinst  eqintqnt  normal
randomised_block             relatedinst  eqintqnt  normal  eqvar
t_common          twosample               eqintqnt  normal  eqvar
t_separate        twosample               eqintqnt  normal
one_way_aov                               eqintqnt  normal  eqvar
wilcoxon          twosample  relatedinst  eqordqnt
sign_test         twosample  relatedinst  eqordqlt
friedman_aov                 relatedinst  eqordqlt
mann_whitney      twosample               eqordqlt
kruskal_wallis                            eqordqlt
mcnemar_test      twosample  relatedinst  eqdichcat
cochran_q                    relatedinst  eqdichcat
chi_squared                               eqnomcat  chifreq
fisher_exact      twosample               eqdichcat
```

B.8 Shapwilkcoeff.dat

```
.7071 0

.6872 .1677

.6646 .2413 0

.6431 .2806 .0875

.6233 .3031 .1401 0
```

```
.6052 .3164 .1743 .0561

.5888 .3244 .1976 .0947 0

.5739 .3291 .2141 .1224 .0399

.5601 .3315 .2260 .1429 .0695 0

.5475 .3325 .2347 .1586 .0922 .0303

.5359 .3325 .2412 .1707 .1099 .0539 0

.5251 .3318 .2460 .1802 .1240 .0727 .0240

.5150 .3306 .2495 .1878 .1353 .0880 .0433 0

.5056 .3290 .2521 .1939 .1447 .1005 .0593 .0196

.4968 .3273 .2540 .1988 .1524 .1109 .0725 .0359 0

.4886 .3253 .2553 .2027 .1587 .1197 .0837 .0496 .0163

.4808 .3232 .2561 .2059 .1641 .1271 .0932 .0612 .0303 0

.4734 .3211 .2565 .2085 .1686 .1334 .1013 .0711 .0422 .0140

.4643 .3185 .2578 .2119 .1736 .1399 .1092 .0804 .0530 .0263 0

.4590 .3156 .2571 .2131 .1764 .1443 .1150 .0878 .0618 .0368 .0122

.4542 .3126 .2563 .2139 .1787 .1480 .1201 .0941 .0696 .0459 .0228
0

.4493 .3098 .2554 .2145 .1807 .1512 .1245 .0997 .0764 .0539 .0321
.0107

.4450 .3069 .2543 .2148 .1822 .1539 .1283 .1046 .0823 .0610 .0403
.0200 0

.4407 .3043 .2533 .2151 .1836 .1563 .1316 .1089 .0876 .0672 .0476
.0284 .0094

.4366 .3018 .2522 .2152 .1848 .1584 .1346 .1128 .0923 .0728 .0540
.0358 .0178 0

.4328 .2992 .2510 .2151 .1857 .1601 .1372 .1162 .0965 .0778 .0598
.0424 .0253 .0084

.4291 .2968 .2499 .2150 .1864 .1616 .1395 .1192 .1002 .0822 .0650
.0483 .0320 .0159 0

.767 .748 .762 .788 .803 .818 .829 .842 .850 .859 .866 .874 .881
.887 .892 .897 .901 .905 .908 .911 .914 .916 .918 .920 .923 .924
.926
```