LAND INFORMATION SYSTEMS

AN OVERVIEW AND OUTLINE OF SOFTWARE REQUIREMENTS

V J S Price

Submitted to the University of Cape Town
in fulfillment of the requirements for the degree of
Master of Science in Surveying

Cape Town

July 1990

by V J S Price
June 1990

Declaration by Candidate:

I hereby declare that this thesis is my own work and that it has not been submitted for a degree at any other University.

V J S Price

Cape Town

2nd July 1990

Acknowledgements

I Should like to like record my thanks to all those who helped me with this study; in particular, my supervisors Professor Heinz Ruther and Mr Colin Martin; Mr Chan Wing Kong of Siemens, Singapore; Mr Douglas Milne of the Municipality of Cape Town; and Mr Derek Clarke of the Directorate of Surveys and Mapping, Mowbray.

Synopsis

This thesis looks at some aspects of land information systems. The introduction gives the rationale for this study, and the second chapter outlines the development of land information systems with particular reference to the cadastre.

In the third chapter the software requirements for the development of land information systems are considered. Programming language and databases are discussed.

The fouth chapter deals with the organisation and hardware needed for a land information system.

Finally, in the fifth chapter some of the algorithms used in land information systems are presented.

Four appendices cover the programmes which were developed in the course of this study, the software specification for an operational system, an example of LIS-related data in a large organisation, and the syntax of Modula-2, the programming language used for the examples.

TABLE OF CONTENTS

1. INTRODUCTION

The Research Information Advisory Group (RIAG) set up by the Land Surveyors' Division of the British Royal Institution of Chartered Surveyors has published (February 1988) a report approved by the Divisional Council setting out the Divisional research policy[1]. In this report the Division sees two roles for itself, firstly to identify areas in which research will be of future importance to RICS members, and secondly, to encourage research in these areas. There was seen to be a need for research in four areas -

i). land surveying activities in land and property development

ii). global positioning systems

iii). land information systems

iv). close range measurement

The first three areas are closely connected. Global positioning provides a framework on to which land information can be tied. Much land information is at present based on 'local origins' (or disconnected reference systems) even in highly developed countries. Such information can only form part of a coherent national land information system if its relationship to other parts of the country can be established. In the same way aerial photographs and satellite images can only be integrated into a land information system when they can be referenced to reliable ground control. Good development planning requires that as much information as possible about the development area is made available as

---

1 Research Advisory Information Group, Land & Minerals Surveyor (L & MS) February 1988.

quickly and as cheaply as possible. This is the role of the land information system. As the development and management of land information systems occupies a central position it is likely that it will become a core activity for land surveyors in the future. The Surveyor as a manager of land information will be responsible for storing data obtained by traditional surveying methods such as photogrammetry, remote sensing, and tacheometry, as well as data from other sources such as soil surveys, land-use surveys, demographic surveys etc. He will also be responsible for analyzing the data and presenting it to those planning and undertaking development.

This is the rationale behind this study of some aspects of land information systems. The emphasis will be on the use of PC's because in all probability the demand from the private sector will be for PC-based systems. These, while not able to handle large volumes of data, can down-load data from large central databases as required for particular projects.

The thesis deals firstly with the development of land information systems and the current applications for such systems. Next some modern software development methods which may have application in the field of LIS are reviewed. After this some of the terms and concepts used in LIS are explained, and, finally, algorithms and software are developed to implement some of the concepts outlined previously.

2. LAND INFORMATION - HISTORY & OVERVIEW

Land Information systems cover such a wide variety of applications and contain such diverse data that they are difficult to categorise. The terms Geographical Information System (GIS) and Land Information System (LIS) are often used interchangeably. At a meeting in Montreux in 1981 the FIG defined an LIS as A tool for legal, administrative and economic decision making and an aid for planning and development which consists on the one hand of a data base containing spatially-referenced land-related data for a defined area, and on the other hand, of procedures and techniques for the systematic collection, updating, processing and distribution of the data. The base of a land information system is a uniform spatial reference system for the data in the system, which also facilitates the linking of data within the system with other land-related data.

This definition covers systems which are called 'land information'(LIS) and those called 'geographic information' systems(GIS). In fact though, the terminology is usually discipline-based. As a generalisation it may be said that a GIS is usually concerned with natural resources, environmental, or demographic data which is presented at small scale and is analysed on a grid cell basis. LIS, on the other hand, is usually concerned with large scale land administration data and is based on polygons which represent land parcels[1]. This distinction is by no means rigid. In the 'multi-purpose' cadastre natural resources, environmental, and demographic data are linked to a parcel-based system; some administrative systems use raster graphics, while environmental systems may be polygon-based. Part of the problem of classification is that no two systems are the

---

1 Hamilton A C and Williamson I P, A critique of the F.I.G. definition of "Land Information System", F.I.G. International Symposium on LIS, Edmonton, 1984.

same. Rhind[2] states there is no dichotomy of the world into two ranges of map scales. In this thesis the term **'land information system'(LIS)** will be used to cover all classes of spatial information systems.

In functional terms two types of land information system may be distinguished:

Systems integrating data from several organisations (Open systems)

Systems using data internal to the organisation (Closed systems)

The first type is embodied in a national land information system or 'multi-purpose' cadastre. Many examples of the second type are found in development agencies.

On a **technical** level the differences are of scale rather than kind. National systems may be required to handle millions of land parcels and handling such large databases provides a great challenge to existing database technology.

A more significant difference is on the organisational level. Personal and political problems inevitably arise when data has to be obtained from a number of different organisations and much of the effort in setting up a system goes into resolving these problems. Note that the open system is not limited to 'cadastral' or parcel based systems. The Natural Research Environment Council (NERC) in the U.K. is an umbrella body which comprises such statutory bodies as the British Antarctic Survey, the Geological Survey of Great Britain, and the Institute of Oceanographic Sciences. This

---

2 Rhind DW,Geographical & Land Information Systems: Their Relationship, Shortcomings and Future Prospects, Conference of Southern African Surveyors, March 1989.

Council recently carried out a study[3] into the feasibility of a common LIS for NERC and into the implementation of such an LIS. This serves to illustrate that integration is taking place in the field of scientific applications as well as in the legal field.

2.1 The Cadastre

'Cadastre' is the technical term used to describe a system in which a legal record of property holdings is combined with a survey identifying the location of those holdings.

Rights in land have been recorded since men first gave up the nomadic life and settled as farmers and town-dwellers. Land is immovable and varies in quality and suitability for the many uses to which it is put. Disputes over land have often proved to be cause for war as well as lying at the root of many family feuds. To avoid such disputes therefore, it has proved necessary to demarcate the boundaries of land holdings and to record the rights of the owner or occupier in the land.

It is extremely rare for an individual owner to possess the absolute right to do as he pleases with his land. He can exercise no more rights than are given to him by the State which for its part guarantees his ownership through the maintenance of law and order.

Egyptian writing almost five thousand years old refers to the registration of property rights. Later documents from the Ptolemaic period give much information about the measurement, registration and quality of land[4] .

---

3 Rhind DW & Green NPA, Design of a Geographical Information System for a heterogeneous scientific community, Geographical Information Systems, Vol.2 No.2 June 1988.

4 Dowson & Sheppard, Land Registration, HMSO 1952.

The Romans learnt a much from the Egyptians about land management. Cadastral surveys for the purpose of levying land taxes were undertaken in various parts of the Roman Empire, property registers were kept, and surveys were regularly carried out for the laying out of towns[5].

The Chinese Empire separated from the Middle East by thousands of miles nevertheless developed a system for surveying and registering property rights.

Likewise surveys for revenue purposes were carried out in Southern India in the eleventh century[6].

These examples serve to show that information about land has been regarded as very important from ancient times. In the past this information was used chiefly for taxation and for the avoidance of disputes over ownership.

In modern times the pressure of increasing population has forced states to curtail the rights of owners in their land. e. g. to control land use so as to achieve a reasonable quality of life for all. Effective control requires knowledge. Planning is only meaningful when the existing patterns of land use, ownership and resources are known.

How can a cadastre be defined? A brief definition was given above, but to quote Dale "A cadastre is a general, systematic and up-to-date register containing information about land parcels including details of their area, value and ownership."[7] This definition is deficient in that it does not

---

5 A Course of Lectures Prepared for Third Year Survey Technicians (Cadastral) Studying for the National Diploma Examination, Vol.1 History and Legislation Relating to Land Surveying in South and South West Africa, Director General Surveys, Mowbray, 1961.

6 Dale P F, Cadastral Surveys within the ~~monwealth~~ Commonwealth, HMSO, 1976.

7 ibid, p1

mention a key element - the geographical position of the land.

However much information there may be about a piece of land, it is worthless unless the position of the land is known. In the early days of British administration in Sabah (North Borneo) land grants were made in the form of "Provisional Leases". Such leases were for a specified area of land, the boundaries to be determined subsequently by survey. In some cases the survey has never been carried out and it is obvious that there is not sufficient unallocated land in the district to cover the extent of the grant. Nevertheless such leases are bought and sold (and registered and transferred) but at a very substantial discount on the price paid for surveyed land. A similar example may be quoted from Botswana - A block of farms in the Ghanzi District was allocated in 1891 on the basis of 'certificates of occupation' for 5000 morgen. Survey of the holdings was delayed until the 1950's with the consequence that the surveyor had a very difficult task to reconcile actual occupation with the area granted. Such cases illustrate that registration without survey has very little value. This fact is plain to land surveyors but unfortunately not always obvious to land administrators, otherwise the cases cited above could not have arisen.

The definition quoted above could be improved as follows "A cadastre is a general, systematic and up-to-date register containing information about land parcels including details of their **location**, area, value and ownership."

2.2 The "Multi-Purpose" Cadastre

The term **multi-purpose cadastre** has been used by Dale and others to mean a land information system which is based on a cadastral survey. The terms 'lot-based system' and 'parcel-based system' are also used. **Multi-purpose cadastre** is a particularly apt name because it indicates that the system is based on, and has grown from the cadastral survey.

There is a vast quantity of information about land not directly related to ownership rights or valuation e.g. land use, soil type, crop, geology, buildings, occupants... Some of this information is of a public nature in the sense that it is necessary for proper planning of the environment and for the provision of services. Many authorities are involved in the provision of services and each has a tradition of independently collecting and storing the information it requires to fulfil its function. This has led to much duplication of data collection and because of the fragmented nature of the data, users may be required to conduct searches for data in several offices. According to Fletcher[8] in the early 1970's 700 000 man-hours per year were spent in New South Wales in searching for data relating to the dimensions and title of property. He estimated that this figure could be doubled if data relating to minerals, forestry, value and services were added.

There is a general realisation that by integrating such data with land title information in order to form a "multi-purpose cadastre" very substantial benefits can be obtained in terms of productivity and better management of resources. However the difficulties in realising such a multi-purpose cadastre are enormous and mainly political. According to Hawkey

"The major difficulties are not those of technical development, for indeed technology is advancing faster than the ability to fully utilise it, but lie rather in the political arena, in organisational and administrative structuring and restructuring, and in agency and people attitudes.....

The process of implementing a national, integrated land information system bears little relationship to those

---

8. Fletcher L N, The Integrated Survey Systems and Modern Mapping in New South Wales, Proceedings of URPIS-ONE, 1973.

involved in the establishment of a single theme system, or a localised, composite database. The difference lies in the magnitude and complexity of the task. This requires the co-ordination and co-operation of many agencies and people with diverse roles and ambitions, the acceptance of rationalisation in the national interest, the assessment of universal users and the solution of practical data integration problems."[9]

It is in the area of building a national land information system that the greatest challenge arises, the challenge to get different departments to co-operate and share data and to find ways of handling the vast bulk of the data.

2.3 Environmental Data

Much scientific work, particularly in the natural and social sciences, produces spatially referenced results. For example study of fauna is closely connected with study of habitat. This in turn is linked to climate, soil type and land form. Climate is determined by local as well as continental and oceanographic factors, while soils are a product of climate and the underlying geology of an area. The net spreads ever wider encompassing all aspects of the environment. Because of the interrelation of so many factors the scientist needs data in all these areas. The land information system provides the ideal way to provide him with this information.

Another example - An agriculturalist is considering developing an area for a particular crop. He must consider such diverse factors as availability and skill level of workers, distance from markets, communication links, climate, soil, slope, water resources, population density and distribution of pests, existing land use and occupation etc.

---

9 Hawkey W N, Land Information New Zealand: The Keys to Progress, Regional Seminar on LIS in Pacific Rim, July 1987.

Gathering this information for a new development project is a formidable task, and unfortunately, in the past the results of such surveys have not been available for the evaluation of other projects in the same area. However, if this data can be put on to a LIS, a comprehensive picture of the region or country will be built up with the passing of time. The major difference from the past is that the information can be made readily available to everyone else who needs to use it.

It follows that much time and money can be saved when data is readily available. This gives the data a commercial value. Sales of such data can provide the necessary economic foundation on which to build environmental land information systems.

2.3.1 Information Technology and the Computer Age.

Computers make it possible to handle much more data than was possible before. With manual systems of data storage strong-rooms could be filled with files but extracting meaningful information from those files was very time-consuming. Using computer technology much larger volumes of data can be stored. The computer also makes it possible to analyse the data. The orderly development of modern society requires more data about land than could be kept under the old systems, - the computer has provided the tool to handle that data, hence the great demand for land information systems amongst all concerned with land administration and development.

Land Information Systems began more than 20 years ago. The Swedish system which is looked on as a model by the rest of the world started at that time. The State of Sabah in East Malaysia also took a decision in 1967 to computerise its land register[10]. In the early seventies many local authorities

---

10 Hj Zainal Abidin, Progress and Problems of Computerisation of Land Administration in Peninsular Malaysia, Regional Seminar on LIS in Pacific Rim, July 1987.

around the world were computerising their land records to form 'land data banks'. Private surveyors have also slowly been computerising their records so that now a great deal of survey information is available in digital form. This has engendered very little excitement because standard database management techniques pioneered in the commercial sector are used. i.e. commercial database management software such as dBaseIII is used to store lists of co-ordinates.

2.4 Automated Cartography

Since the early 1960's the use of computers in map production has been continuously developed. Computers have been used to store map data and to drive plotters to produce paper maps. The focus in this work has been the production of paper maps, not the creation of a digital spatial database. It is this difference in purpose which distinguishes what may be called "automated cartography" from the spatial database which lies at the heart of an LIS.

2.5 DBMS and Computer Graphics

Work from three separate directions is coming together - automated cartography, computerisation of cadastral records, and the statistical analysis of spatial data, fields involving geodesists, photogrammetrists, cartographers; administrators, lawyers, cadastral surveyors; environmentalists, foresters, agriculturalists, planners etc.

Development has been "application driven" i. e. it has been aimed at solving real problems rather than growing from an established body of basic research.

Applications vary in different countries depending on need. Until now the emphasis has been on large systems using large databases but the increasing power and falling prices of PC's are beginning to make the development of PC based systems feasible e. g. for farmers to track input of chemicals and

output of crops on a field by field basis, to record the spread of pests and diseases etc.

3. SOFTWARE DEVELOPMENT

3.1 Spatial Data

Spatial data lies at the centre of land information systems. It is this type of data which distinguishes them from more familiar data base systems such as those which maintain details of bank accounts or electoral rolls. Spatial data provides information about the position or location of an object. This may be a point object such as a well or triangulation station, a linear object such as a road or railway line, or an areal object such as a farm or the extent of a specific soil type.

Before the advent of computers the most common method of storing spatial data was the paper **map**. Points, lines, and areas are represented on maps by a scale drawing of the actual features. Attribute data may be written on the map, alternatively shading, hatching, and symbols may provide a reference to a legend on the map. When more extensive attribute data is involved a map reference may provide a reference to an entry in a land register or a survey record file.

3.1.1 Representation of Spatial Data

There are two ways in which spatial data can be represented. Land areas may be depicted by either vector or raster (grid) data. Hitherto these methods have been regarded as competing, but the present view is that they are complementary and that each has its place in storing spatial data.

3.1.1.1 Raster Data

The simplest raster data structure consists of an array of grid cells or picture elements. Each array element contains a value which represents either the attributes of the corresponding point in space or a pointer to a data file containing the attribute. The position of each grid cell in

the array corresponds to its position on the ground. In the case of a screen display, each grid cell maps to a single point on the screen and also to a square on the ground[1]. (By adopting a larger scale, the grid cell can be mapped to a block of pixels rather than to a single pixel. However, nothing is gained through this; on the contrary, the display simply becomes harder to interpret.) The same mapping applies between the grid cell and a paper plot of the data.

The accuracy which can be obtained depends on the resolution at which the data can be displayed. For example, a high-resolution screen might have 40 pixels (picture elements) per cm. If a map is to be displayed at a scale of 1/10000 each pixel will represent a square of 2.5x2.5 m. Furthermore the file will occupy about 25 x 40 x 25 x 40 x b where b = No. of bits required for colour or attribute data. This file will obviously be several megabytes in size. It may appear from this that use of the raster method of data representation is not practical because of the large files needed and the relatively poor resolution obtained. However, because large numbers of points will inevitably share the same attributes, file compression algorithms can make the files substantially smaller. Commercially available data obtained from satellites may have at best a resolution of 10m x 10m therefore for this type of data a higher resolution in the computer is meaningless. There are many types of spatial data which have "fuzzy" boundaries. For example, the boundaries between different soil or vegetation types extend over many metres and cannot be located precicely to a centimetre or even a metre. Raster storage is well suited to such data.

---

1 Burrough P A, Principles of Geographical Information Systems for Land Resource Assessment, Oxford University Press.

3.1.1.2 Vector Data

In some cases accuracy of representation is important. A characteristic of land which is of interest to all land owners is its area or two-dimensional extent. In this case a storage method must be adopted which will preserve the full survey accuracy, whether to the nearest centimetre or milimetre. This can best be handled by using two dimensional co-ordinate geometry. Where large areas are concerned, appropriate projections can be used to map the actual land area on to a plane surface. Land areas can also be represented (more accurately!) by spheroidal co-ordinates or three-dimensional rectangular co-ordinates but at a cost of greatly increased processing time.

Each area of land, mapped on to a plane, may be represented by a polygon provided that curvilinear boundaries are replaced by a series of straight lines. If the straights are sufficiently short the boundary will approximate very closely to an actual curvilinear boundary.

It should also be noted that most features commonly represented by lines and points are not actually lines and points but very small polygons and very thin polygons e. g. a well or borehole has a finite area, a powerline has a finite width respectively.

The polygon is a compound object - it is bounded by lines. Each line has a point at both ends. The point is thus the fundamental element in spatial analysis, analogous to the atom in nineteenth century physics. A line is defined by an ordered pair of points, and a polygon by an ordered set of lines.

The representation of spatial data by points, lines, and polygons is commonly called **vector** representation, though it has little to do with the conventional meaning of the word vector.

3.1.1.3 Computer Representation of Spatial Data

In older programming languages (FORTRAN, GW-BASIC) no matter what conceptual model is adopted for the data, it still has to be modelled in terms of numbers, characters, and arrays of numbers in order to be implemented on a computer. In other words the programmer is serving as a translator between the conceptual model and an intermediate implementation model. This is wasteful of effort and increases the risk of programming errors. Modern programming languages, especially 'C', Pascal, Modula-2 and Ada, permit the creation of a wide range of user-defined data types which can follow the conceptual model of the data. Modula-2 and Ada have the advantage that the structure of the defined data types can be hidden from the programmer. A further advantage offered by the new languages is the use of dynamic abstract data types. The list, stack, queue, and tree are the main varieties. These data objects can be created and erased and can grow and shrink during the execution of the program.

3.2 Attribute Data

Spatial data on its own (i.e. sets of points, lines, and polygons) does not convey any useful information. To make spatial data useful the meaning of each spatial element in the database must be described by its attributes. For example, a line in a spatial database might have attributes which define that line as a gas-pipe line, 20 cm diameter, made of polystyrene, constructed in 1985 in a right-of-way registered as Servitude No.1352/84, depicted on the map as a 0.5 mm yellow line, etc. This data may be very extensive and it would present severe problems to include it in the same data base as the spatial data. In many cases too, this attribute data already exists in non-spatial ~~compter~~ computer databases - in the development of an LIS the two have to be brought together.

Another important example of attribute data is the street address. To the surveyor people may live in 'polygons' (erven or sectional title units) but to the man in the street the key to his locality is his street address. Remarkably few people know the number of the erf on which they live but everyone knows his address. Thus in a practical LIS it is important that addresses as well as erf numbers can be used as a key to the database.

3.2.1 Classification of Data

Many relationships exist between different data elements. Some of these may (or may not) be obvious to a human but are not easy for a computer to detect, e.g. we can recognise that a shop is a type of commercial premises, or that a footpath is a kind of right of way. To make these relationships easy for the computer to detect, a classification and coding system must be adopted. Thus hierarchical coding systems for map elements, land use, planning zoning etc. are used. These systems group togther objects which belong together.

3.3 Databases

When data is put into a computer the often unspoken assumption is that it will be possible to retrieve that data within a reasonable time (if a computer takes thirty minutes to produce information which could be obtained in five minutes by finding and opening a file, then the computer is reducing efficiency) and to make deductions from that data by applying logical operators to the data. The type of queries and the expected response time will be the major considerations affecting the way in which the data is stored,- in other words the design of the data structures. It must also be kept in mind that the 'conceptual' or 'logical' data structure (how the programmer sees the data) need not necessarily be the same as the 'physical' data structure (where and how the data is actually stored on disc and in memory). The heart of a LIS system is the data query

mechanism, and an appropriate solution to this problem is the key to a successful LIS.

A database may be defined as "a collection of inter-related data stored together with controlled redundancy to serve one or more applications in an optimal fashion; the data are stored so that they are independent of the programs which use the data; a common and controlled approach is used in adding new data and modifying and retrieving existing data within the data base."[2]. In this definition "controlled redundancy" means that redundant or duplicated data must be kept to a minimum consistent with efficient access to the data, and that alterations to such redundant data must be carefully controlled to ensure that the duplicate copies of the data are always consistent. The essence of a database is that different applications may use the same data files. e.g.files of clients, suppliers, stock inventories, debtors' ledger etc may be used for an order system, a stock control system, or an accounting system.

The above definition applies equally to spatial and non-spatial data. However, because databases were originally used for non-spatial data, the term is sometimes limited to this type of data. The LIS is the integration of databases of spatial and related non-spatial data.

Data only becomes useful when it is viewed in relation to other data, for example, a list of names is meaningless unless more data is added to it such as addresses, telephone numbers, jobs etc. A part number means nothing unless we know what part the number refers to, who makes it, what the price is, how many units are in stock etc. In real applications the relationships between data items may be quite complicated and the database system must indicate these relations.

---

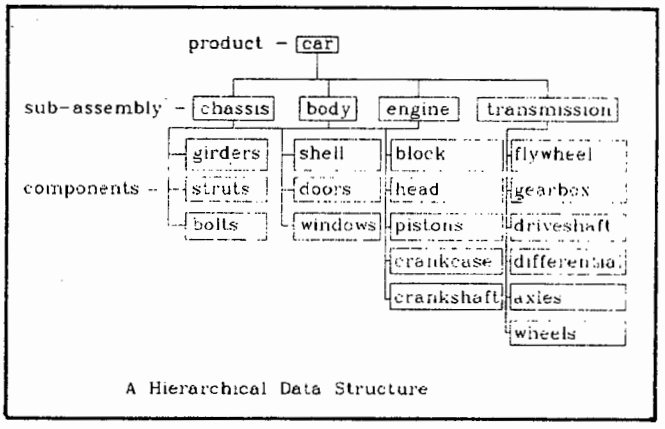
2 Martin J. Principles of Data-Base Management, Prentice-Hall 1976.

3.3.1 Classification of Databases

Database systems may be categorised by the method used to relate the various data items. There are three broad types[3].:-

3.3.1.1 Hierarchical

The hierarchical type represents data relations as a hierarchy or tree structure. The data is divided into parent-child relations where each parent may have many children but each child has only one parent.



A Hierarchical Data Structure

Fig. 3 - 1

---

3 Hughes J G, Database Technology A Software Engineering Approach, Prentice-Hall 1988.

3.3.1.2 Network

Network databases express more complex relations. The structure is similar to the hierarchical type but a child may have many parents.

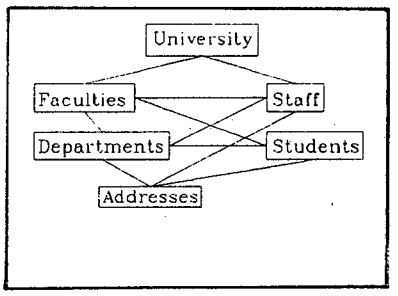


Fig. 3 - 2 A Network Data Model

3.3.1.3 Relational

The **relational database** model was first developed by E.F. Codd in 1970[4]. It is important to note that the relational model relates only to the **logical** or user's view of the data. It is not feasible to construct a large **physical** database using the relational model. The logical view of the data is how it appears to the user or programmer. This is not necessarily how the data is stored in the computer e.g. a sparse matrix or spreadsheet appears as a matrix to the user but is normally stored in memory as a linked list. Such physical storage is transparent (invisible) to the user.

The relational model places all data in two-dimensional tables or rectangular arrays. These tables have the following properties:-

---

4 Codd E F, A Relational Model of Data for Large Shared Data Banks, Comm. ACM 13 (6/6/70) p.377-387.

Each row in the table is unique. Duplicate entries are not allowed.

The data in any column is all of the same kind e.g. one column might contain names, while another contains addresses.

The sequence of rows and columns can be changed without changing the information content of the table. Any function using the table is independent of the order of the table.

Each row of a table is called a **record** or **tuple**. Each data item in a record is called a **field**. One field, or the combination of two or more fields, forms a **key** to the record, that is, it provides a reference by which the whole record can be located. For example, the 'name' field might form a key to access a table consisting of names and addresses.

Relations between data items are facts e.g. a name may be associated with an address, a student takes a particular course, etc. However the pattern of relationships may be seen in different ways. The data used in a particular context may be perceived to be hierarchical or networked from the everyday viewpoint; however, it is possible to convert all data structures to the relational form through 'normalisation'. The primary aim of normalisation is to remove all data redundancies. The importance of this is that a relational database can be used no matter how the data is structured.

3.3.2 Security

Every organisation has confidential information which could harm the organisation if it fell into the wrong hands. e.g. a company's profits might be reduced if information on product development or marketing strategy was divulged to a competitor. Labour unrest might result from confidential

reports on staff becoming known to the staff concerned. At the same time employees need a lot of the information in the company's database to do their jobs efficiently. There is thus a need to hide some information from certain people while making it available to others.

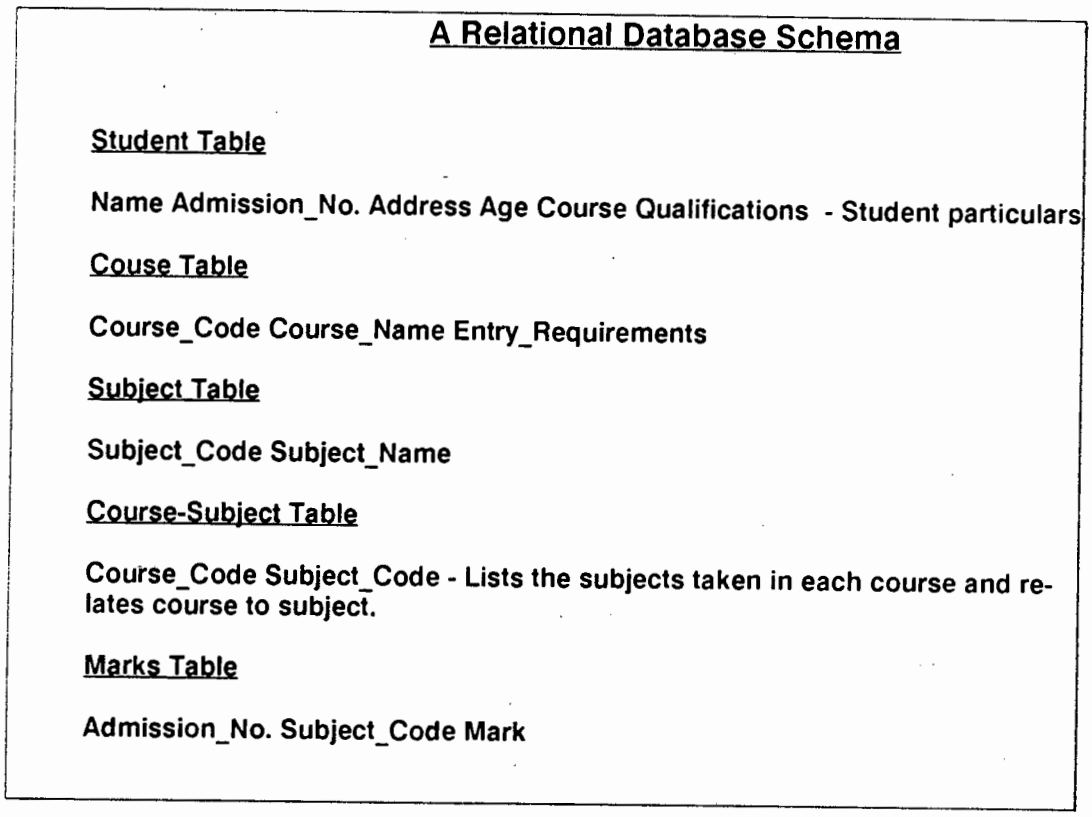


Fig. 3 - 3 A Relational Database Schema

Damage could also be done to an organisation by the careless or malicious alteration of data even if that data is not

confidential. Computer operating systems (other than those on stand-alone micro-computers) use a system of passwords to restrict access to files. Commonly the owner of a file can give rights to other users to read and/or write to files, to create new files, to delete files or to search for files within a specific directory. He can also delegate the right to others to grant the rights they have to third parties. If a specific user does not have the right to search for files within any directory, that directory will appear empty to him although it actually contains fies. This type of access control is not good enough for database access control because it may be necessary to restrict access to certain fields within a file. e.g. a file may contain erf numbers, area, registered owner, owner's address, date of acquisition, purchase price, and deed number. It may be a legal requirement that the purchase price should not be available to the public, but that access to it should be restricted to certain authorized officials. It may also be a requirement that alteration of data can only be carried out by one or two senior officials. Modern databases permit such controlled access and allow access rights to be granted to individuals or groups of individuals.

It is one thing to have a system which provides the tools for security but another matter to implement it effectively. Even when the system is set up properly security still depends on the right attitude on the part of the users of the system i.e. not to give their passwords to others, not to leave their terminal logged in and unattended etc.

3.3.3 Query Languages

A **relational calculus** has been developed to formulate queries to relational databases. This gives a theoretical basis to operations performed on the data and enables the correctness of such operations to be proved. For this reason current

interest in the data processing world is focussed on relational databases which have been described above.

In 1971 Codd followed up his relational model for databases with the design of a query language to access such databases[5].

Since then many variations on the original relational query language have been developed. One of these, SQL (Structured Query Language) is becoming established as a de facto industry standard for relational databases and for this reason will be expanded on below.

3.3.3.1 SQL - A Database Query Language

A consequence of the publication of Codd's ideas on relational databases was that a great deal of experimentation into ways of implementing such databases took place. A number of query languages were produced in the early 1970's. One of these languages, developed by IBM and released in 1975 was called SEQUEL ("Structured English QUery Language"). In 1977 a revision called SEQUEL/2 was released. This was subsequently re-named SQL. In 1983 IBM brought out DB2 which is broadly compatible with SQL. Meanwhile other vendors picked up SQL, and by 1986 more than 50 products on the market supported some dialect of SQL. These include 'Oracle' which was SQL-based from the start, 'dBase IV' which maintains compatibility with 'dBase III' (a popular database programme for personal computers) as well as having an SQL interface. Thus SQL has become a de facto standard. It has also become a de jure standard. A version of SQL has been adopted by the American National Standards Organisation (ANSI) as a standard relational language. This is likely to be adopted by the ISO in the near future.

---

5 Codd E F, A Data Base Sublanguage founded on the Relational Calculus' Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control.

Without going too deeply into the matter an outline[6] of the main SQL commands will be given as they illustrate the workings of a relational database.

It was mentioned above that relational databases store data in two dimensional tables. There are two types of tables - 'base tables' which represent actual physical data storage, and 'views' which are virtual tables i.e. they are created temporarily from data in other tables for display, data entry, print-out and updating but do not reflect actual relations between data elements in the physical database. The first step in creating a database in SQL is the command

```
CREATE SCHEMA AUTHORIZATION XYZ
```

In everyday usage a data 'schema' means the tables of relations which comprise the database, but it is more accurately defined as the part of the database owned by a specific user. A complete database may comprise a number of schemata each with a different owner who may control updating, say, of his schema. The words 'AUTHORIZATION XYZ' mean that XYZ is the owner of the schema. Other users can only access data in this schema in so far as XYZ grants privileges to them.

Having created a schema, tables are created with the 'CREATE TABLE' command e.g.

```
CREATE TABLE MEMBER (SURNAME   CHAR(20) NOT NULL
                     INITIALS  CHAR(10)
                     TITLE     CHAR(6)
                     ADDR1     CHAR(30)
                     ADDR2     CHAR(30)
                     ADDR3     CHAR(30)
```

---

6 Date C J, A Guide To The SQL Standard,Addison-Wesley, 1987.

```
                    COUNTRY  CHAR(20)
                    POSTCODE CHAR(10))
```

The expression **'NOT NULL'** following the field 'SURNAME' means that this field must have a value. If one tries to insert a record without a surname it will not be added to the database. In addition to character data, several types of numeric data fields are permitted.

There are four basic data manipulation commands - **SELECT**, **INSERT**, **UPDATE** and **DELETE**. These operate as follows:-

```
INSERT INTO MEMBER (SURNAME, INITIALS, TITLE, ADDR1,
                    ADDR2, ADDR3, COUNTRY, POSTCODE)
VALUES('Simpson','T','Mr','Blk 552 Jurong West St.21'
,'#04-221','Singapore','REPUBLIC OF SINGAPORE','0417')

UPDATE MEMBER
SET POSTCODE = '0418'
WHERE MEMBER.SURNAME='Simpson' AND MEMBER.INITIALS='T'

DELETE FROM MEMBER WHERE MEMBER.POSTCODE = '0418'
```

The **'SELECT'** command creates new temporary tables which may be another complete table, a sub-set of another table, or the result of a relational join operation between two tables which have a common column. e.g.

```
SELECT * FROM MEMBER WHERE COUNTRY='SOUTH AFRICA'
```

The asterix means that all the columns of the table 'MEMBER' are selected, the alternative is to give a list of column or field names. 'WHERE COUNTRY="SOUTH AFRICA"' means that the new table will contain only the records in the main table which have 'SOUTH AFRICA' in their 'COUNTRY' field.

Despite its general acceptance as the standard query language for commercial databases some writers have found it to be inadequate as an LIS query language specifically in that it lacks a 'loop' construct and recursion[7].

3.3.3.2 Artificial Intelligence

The terms 'artificial intelligence' (AI) and 'expert system' are to some extent used interchangeably whereas strictly 'expert systems' are a branch of 'artificial intelligence'. Research in this area attempts to get a computer to apply the same logical reasoning processes to a problem as the human expert applies.

AI has an important role to play in LIS in two areas. The first is to provide a natural language interface between the LIS and the user. Most people who need information from an LIS are not system experts. At present they rely on the expert who can translate their queries into a form understandable to the computer and then get the answer to their query. However experts are in short supply. To enhance the value of an LIS, and thus improve its cost/benefit ratio, the more people who use it the better. This means making the system easier to use so that a query could be given in plain English (or any other language) and the AI component of the software would translate this into the appropriate database query language. The AI program would also ask the user questions to clarify any ambiguities in the original query.

The second area in which AI has a role to play is in the analysis of the data contained in the database. To do this the 'expert system' may be used. A human expert solves a problem by applying his book knowledge and knowledge gained by experience to the problem and making logical deductions

---

7 Frank A U,LOBSTER: Combining Database Management and Artificial Intelligence Techniques to Manage Land Information, Proceedings of FIG XVIII International Congress,Vol.3 pp.2-13, Toronto 1988.

until he arrives at the solution. Very often the reasoning may be carried out sub-consciously or intuitively.

The expert system comprises the 'knowledge base' and the 'inference engine'. The knowledge base contains the expert's knowledge expressed in the form of rules like 'if X then Y' or 'if X and Y then Z'. The 'inference engine' is a program module which applies the theorems of prepositional logic to the knowledge base in order to produce an answer to the initial query.

Expert systems have been linked to LIS to assist in mineral exploration, and to evaluate well sites for need of protection against pollution[8].

PROLOG is a computer language which is based on predicate logic. For this reason it is widely used in artificial intelligence applications. Prolog is well suited for use as a database query language and a version of Prolog,called 'LOBSTER', has been developed for use as an LIS query language[9]. PROLOG belongs to a class of programming languages called **functional** languages, in contrast to more familiar **procedural** languages such as FORTRAN, Modula-2, BASIC etc. Functional languages, while suited to AI applications, are general purpose and may be used for most programming tasks.

3.3.4 Object-Oriented Data Bases

Commercially available relational databases at present can only handle simple data types such as numbers, character strings, and dates. They are therefore not able to deal

---

8 Uhlenbruck H M, The Application of Expert Systems in Geographical Information Systems, University of New Brunswick Department of Surveying Engineering Technical Report No.138, June 1988.

9 Frank A U,LOBSTER: Combining Database Management and Artificial Intelligence Techniques to Manage Land Information, Proceedings of FIG XVIII International Congress,Vol.3 pp.2-13, Toronto 1988.

conveniently with spatial data. For this reason LIS's are most often hybrid systems combining a relational database holding attribute data with a different storage arrangement for graphic objects. Relational databases which can handle any type of data object are the subject of research and development at the moment and bring **'object-oriented'** programming technology (see below) into the realm of database management. Examples of such languages are RAD[10], Vbase, GemStone, and Statis[11].

Intergraph Corporation have recently announced LIS software called TIGRIS which integrates topologically structured spatial data with an object-oriented relational database[12].

3.3.5 Temporal Databases

The majority of databases are concerned with current information, that is, they must reflect the latest information. In most cases such databases are updated by adding the new information and deleting the old. For example, when a person changes his address his old address is removed from the database and his new one is inserted. The old data may or may not be saved in archive tapes. If it is saved, it is generally troublesome and time-consuming to recover.

In some cases, time forms an important part of the data e.g. in a commercial transaction ordering, despatch, billing, and payment follow in a logical order and each action forms part

---

10 Osborn S L & Heaven T E,The Design of a Relational Database System with Abstract Data Types for Domains, ACM Transactions on Database Systems, Vol.11 No.3 pp.357-373, September 1986.

11 Rapaport M, Object-oriented data bases: The next step in DBMS evolution, Computer Language, Vol.5 No.10 pp.91-98, October 1988.

12 Marrao E, TIGRIS: A Third Generation GIS, Proceedings of the Ninth Conference of South African Surveyors, Paper No. 9.2, March 1989.

of the overall transaction. It is important to both parties to be able to track the transaction to identify delays in despatch or payment. In almost all commercial database management systems such a transaction is not identified as an object in its own right. Each action is seen as a separate entity having time as an attribute.

There is no doubt that up till now the time element has been neglected in the design of commercial databases. The reasons advanced for this are complexity of the problem[13] and a lack of adequate technology in the early days of database development to tackle it[14].

Since 1980 there has been an increasing academic interest in the development of data models which incorporate time. A recent bibliography[15] listed 80 articles published from 1982 to 1986 on this subject.

Every object or entity has a beginning at some moment in time. It also has a lifespan during which its attributes may change. Finally it may die in the sense that it is purged from the database. A query about the attributes of an object is only meaningful if the time is also specified - for example a person may have the attribute of 'single' in 1985, 'married' in 1989, widow in 2020, and dead in 2025. Note however, that her record in the database will only 'die' when it is removed. Conventional database systems only record the condition of the data object at the instant of the each transaction and so do not give a full view of the objects they model, in fact they represent a 'snapshot' view of the

---

13 Gad Ariav, A Temporally Oriented Data Model, ACM Transactions on Database Systems, Vol.11 No.4 pp.499-527, December 1986.

14 Tsichritzis D C & Lochovsky F H, Data Models, Prentice-Hall, 1982.

15 McKenzie E, Bibliography: Temporal Databases, ACM SIGMOD Rec.15 No.4 pp.40-52, December 1986.

state of the data objects at the current moment. A data model which incorporates time directly therefore, gives a more realistic representation. A general model adopting this view has been developed by Langefors[16] who has proposed what he calls the 'infological model'. In this model the basic element of information may be regarded as a tuple:

name, attribute, attribute value, time

Several implementations have been proposed. In these two main approaches can be distinguished.

The first leaves time out of the data model altogether but includes time as an attribute of the various data entities. The query language includes operators which give the user a temporal view of the data. This approach has been followed by Snodgrass[17] who has developed a temporally based query language. This language which he calls 'TQuel' is an extension of Quel, the Ingress (a well known commercial relational database) query language. It is used to produce a time-oriented view of a conventional Ingress.

The 'computation-tuple sequence'[18] of Ginsberg and Tanaka follow the same direction. An extension of this concept makes use of an 'object-oriented' database[19]. The object to be

---

16 Langefors B, Infological Models and Information Users' Views, Information Systems No.5 pp.17-32, 1980.

17 Snodgrass R, The Temporal Query Language TQuel, ACM Transactions on Database Systems, Vol.12 No.2 pp.247-298, June 1987.

18 Ginsberg S & Tanaka K, Computation-Tuple Sequences and Object Histories, ACM Transactions on Database Systems, Vol.11 No.2 pp.186-212, June 1986.

19 Rapaport M, Object-oriented data bases: The next step in DBMS evolution, Computer Language, Vol.5 No.10 pp.91-98, October 1988.

modelled is constructed as an abstract data type which includes the time element. This object is stored in a relational database which has the ability to handle abstract data types.

The second approach is to include time as an integral part of the data model as is done in the 'temporally oriented data model' of Gad Ariav[20] and in the 'time-relational model' of Ben-Zvi[21].

The temporal element in the cadastral database is dealt with in Sections 3.6 and 5.5 below.

3.3.6 The Requirements for a Spatial Database

As was shown above in 3.1 spatial data has some characteristics which distinguish it from the non-spatial data commonly found in the commercial environment. Thus a database handling spatial data will have some special requirements. Frank[22] has listed some of these as:-

object-oriented database design
generalisation/specialisation available for data
suitable for modelling geometric data
fast access based on spatial location

Object-oriented databases were dealt with in 3.2.3.3 above and object-oriented programming is looked at in 3.4.1 below.

---

20 Uhlenbruck H M, The Application of Expert Systems in Geographical Information Systems, University of New Brunswick Department of Surveying Engineering Technical Report No.138, June 1988.

21 Ben-Zvi J, The Time Relational Model, PhD thesis, Dept. of Computer Science, University of California,Los Angeles, 1982.

22 Frank A U, Requirements for Database Systems Suitable to Manage Large Spatial Databases, Proceedings of the International Symposium on Spatial Data Handling, Zurich, 1984.

3.3.6.1 Generalisation of Map Data

"Generalisation/specialisation" of data means that data objects may have different graphic representations according to the scale the data is being displayed or plotted at. e.g. on a 1/50 000 map a police station may be marked by the letter 'P' whereas at 1/1000 all the walls and fences of the station will be shown. Aggregation of data is also required at small scales e.g. at 1/1000 the individual houses, roads, schools, parks etc making up a town will be shown individually, but at 1/250 000 a hatched polygon on the display would represent the whole town.

At present there is no satisfactory solution to this problem. Some organisations (e.g. Ministry of Public Works, State of Qatar) maintain separate databases for each scale at which they publish maps. This is unsatisfactory because whenever revision is carried out several separate databases must be updated creating unnecessary work with additional opportunities to make mistakes. Furthermore it directly contravenes the principle of minimising data redundancy.

An alternative would be to attach a symbol (or symbols) to each data object, and depending on the scale of the display the object would either be drawn in full or the appropriate symbol would be displayed. Each object could also have an attribute showing to which super-object it belonged e.g. a house belongs to a suburb, a suburb to a town etc. and again depending on the scale either the individual object or the super-object could be displayed.

3.3.6.2 Modelling Geometric Data

To be suitable for modelling geometric data a database must permit the use of variable length records because chains and polygons may have few or many points. Not many relational databases permit this, but as was shown above, relational databases are desirable because of the way in which queries

can be formulated. This is why many current land information systems have a hybrid database - relational for attribute data and a non-relational graphic database.

3.3.6.3 Data Access

"Fast access based on spatial location" means that the database must have an indexing system based on position. Over the past thirty years a great deal of research has gone into the development of efficient indexes for non-spatial data and programmers have a choice of such methods as 'hash' tables, B-trees, AVL-trees etc. (Every standard text on computer data structtures covers these indexing methods.) Much less effort has gone into the development of indexes for spatial data. The problem will be discussed further in 5.2 below.

3.4 Programming Languages

3.4.1 Object-Oriented Programming

Object-oriented programming is a programming style which has emerged from the new discipline of software engineering. Its aims are to cut the cost of software development by promoting the use of re-usable software modules, and by increasing the reliability of software by building large programs from small thoroughly tested components. It is valid for all computer programming.

The concept of object-oriented programming has been admirably summerised by Wiener and Sincovec as follows:-

"No longer is it necessary for the system designer to map the problem domain into pre-defined data and control structures present in the implementation language. Instead the designer may create his or her own abstract data types and functional abstractions and map the real world domain into these programmer-created abstactions. This mapping, incidentally, may be much more natural because of the virtually unlimited range of abstract types that can be invented by the designer.

Furthermore, software design becomes de-coupled from the representational details of the data objects used in the system. These representational details may be changed many times without any fallout effects being induced in the overall software system."[23].

A programming 'object' has been defined as "an encapsulated representation (state) and a set of messages (operations or procedures) that can be applied to that object"[24]. In more familiar terms an object could be described as a data type together with a set of procedures which operate on data of the defined type.

In order to support object-oriented programming style a programming language must have four characteristics:-

Information Hiding

The principle of information hiding is that the internal structure of a program module should be not accessible from outside the module. Data should be passed to and from the module only through a carefully defined interface. This prevents uncontrolled dependencies growing between modules, and enables the implementation of a module to be re-written without affecting other parts of the program. In FORTRAN programs, for example, it is an everyday practice to have a COMMON block which is used to pass global variables to separately compiled sub-routines. The use of these global variables creates complex dependencies between the main program and the sub-routines, and, in general, makes it difficult to use the same sub-routines in other programs.

---

23 Wiener and Sincovec, Software Engineering with Modula-2 and Ada, Wiley, 1984.

24 Thomas D,What's in an Object?, Byte Magazine, p.231 March 1989.

Data Abstraction

An **abstract data type** is typically either a complex data structure or a pointer to such a structure (i.e. the address of the structure). The actual structure of the data element may be private to the module in which it is defined - in this case it is known as an **opaque** data type and forms an example of 'information hiding'.

e.g. A point might be defined in Modula-2 as

```
TYPE point = RECORD
                id : CARDINAL;
                x  : LONGREAL;
                y  : LONGREAL;
             END;
```

Variables may then be defined to be of type 'point' and procedures which operate on data of this type can be defined, for example join, polar, and intersection.

Dynamic Binding and Overloading

It would be useful if abstract data types could be used with different types of data. e.g. a **list** might be required to contain lists of integers or of floating point numbers. A procedure to print the list would require different sub-routines to print each type of list (The internal computer representation for integers is different from the representation of floating point numbers, therefore different steps are needed to convert each type to character strings for printing.). Dynamic binding ensures that when the program is running the correct sub-routine for printing is automatically selected according to the data in the list. e.g. the BASIC PRINT statement exhibits this quality in that it will print strings, integers, or floating point numbers. In Modula-2 on the other hand, separate procedures are needed

to print each data type. This feature is also known as **overloading** where one procedure name or operator is "overloaded" with several functions. e.g. the "+" in BASIC can add integers and floating point numbers and can also concatenate strings, a very different operation from arithmetic addition. Note that while some operators are "overloaded" in nearly all programming languages (e.g. "+" will add floating point or integer numbers), only ADA amongst the more popular languages allows the programmer to create new "overloadings".

Inheritance

The property of inheritance permits programmers to create classes of objects. Objects may then be defined as belonging to a class. The way in which a specific object differs from the general class type may be used to define the object.

Some programming languages have been designed specifically to implement the object-oriented programming concept. The best known of these is **Smalltalk**. However the concept can be implemented to some extent in any programming language because it refers more to a programming style than to a specific language.

3.4.2 Dynamic Data Types

When a program is compiled and linked memory is allocated for **static** variables i.e. those which exist throughout the execution of the program at a fixed address in memory. Any memory space not required by the program and its static variables or the operating system is called the **'heap'**. This is available for the allocation of dynamic variables. The size of the heap will obviously depend on the size of the main memory of the computer. Thus by using dynamic variables it is possible to write a program in such a way that the size of the data set the program can handle is limited not by the program but by the physical configuration of the computer. It

```
     memptr : ADDRESS ;
     memsize : CARDINAL ;
     reply  : CARDINAL ;
BEGIN
memsize := HeapAvail ( MainHeap )-8 ;
 (* 8 for MSDOS relocation *)
HeapAllocate(MainHeap,memptr,memsize);
reply := Lib.Execute(command,params,memptr,memsize) ;
HeapDeallocate(MainHeap,memptr,memsize);
   IF reply = 0 THEN
     Lib.FatalError('Failed to execute program ');
   END ;
END Exec ;

PROCEDURE GetTime(VAR Hrs,Mins,Secs,Hsecs: CARDINAL);
VAR
     R : SYSTEM.Registers ;
BEGIN
     WITH R DO
          AH := 2CH ;
          Lib.Dos(R) ;
          Hrs   := CARDINAL(CH) ;
          Mins  := CARDINAL(CL) ;
          Secs  := CARDINAL(DH) ;
          Hsecs := CARDINAL(DL) ;
     END ;
END GetTime ;

PROCEDURE GetDate ( VAR Year,Month,Day : CARDINAL ;
                    VAR DayOfWeek : DayType ) ;
VAR
     R : SYSTEM.Registers ;
BEGIN
     WITH R DO
          AH := 2AH ;
          Lib.Dos(R) ;
```

```
            Year := CX ;
            Month := CARDINAL(DH) ;
            Day := CARDINAL(DL) ;
            DayOfWeek := DayType(AL) ;
        END ;
    END GetDate ;

    END Diverse.
```

As can be seen the language is very similar to Pascal though there are some important differences. For example Modula-2 is case-sensitive, in other words 'i' is a different variable from 'I'. In this respect Modula-2 is like 'C'. A grammar of the language is given in Appendix D.

3.5.2 Dynamic Data Types in Modula-2

In common with Pascal, Ada, and 'C', Modula-2 has a **'pointer'** data type. A pointer is a variable which holds the address where a value is stored rather than the actual value as is the case with other variable types. The pointer's function is to provide a reference to **dynamic** variables which are created and disposed of while the program is running.

In Modula-2 a pointer variable is declared in this way:-

```
VAR  x : POINTER TO INTEGER;
```

A pointer may be declared to point to any data type. When the pointer is initialised, sufficient space for that particular data type is reserved on the heap. To initialise a pointer the command NEW is used. e.g.

```
NEW(x);
```

This command reserves two bytes on the heap (x is declared as a pointer to an integer, and an integer occupies two bytes), and sets x to the address of the start of the allocated area.

A value at the address stored in x is referred to by writing x^, the '^' is called a de-referencing symbol.

When a dynamic variable is no longer needed it can be de-allocated and the space which it occupied will become available for allocation to another dynamic variable. A variable is de-allocated by the command:-

```
DISPOSE(x);
```

A list can be created as follows:-

```
TYPE
     IntList = POINTER TO IntListElement;
     IntListElement = RECORD
                         data : INTEGER;
                         next : IntList;
                      END;
```

An integer list variable would be defined

```
VAR
     testlist : IntList;
```

Before use the list must be initialised by

```
testlist := NIL;
```

The **NIL** value indicates that the pointer is not referencing any memory location, and in this context it means that there are no elements in the list.

To add the first element to the list a new list element is created by

```
NEW(testlist);
```

The data is placed in the data part of the list element record, and the pointer to 'next' is made **NIL**

```
testlist^.data := data1
testlist^.next := NIL;
```

This can be repeated to add more items to the list. Generally a module will be written to provide procedures to add items to lists, to remove items, to search for items, and to sort lists.

3.6 Temporal Databases and Cadastral LIS

Hunter[25] has clearly pointed out the importance of preserving historical data in land information systems. He has highlighted an important truth, namely that paper records are often preserved simply because it requires an effort to destroy them; magnetic media on the other hand are re-usable and it is all too easy to write new data over the old. For this reason designers of databases need to give serious consideration to what historical data needs to be kept, and how it is to be stored, - optical discs hold great promise in this regard. Unless more attention is paid to this matter historians a hundred years from now may find they have more source material from the nineteenth century than from the second half of the twentieth century!

In some jurisdictions at least, it is not just important to preserve historic data but a legal requirement. Cairns[26] cites cases in which legal problems have arisen through a

---

25 Hunter G J, Non-Current Data and Geographical Information Systems. A Case for Data Retention, International Journal of Geographical Information Systems, Vol.2 No.3 pp.281-286, July-September 1988.

26 Cairns M, The Archival Role in Establishing the Ownership of Land, History of Surveying and Land Tenure in South Africa, Vol.1 pp.89-98, University of Cape Town, 1984.

failure to follow proper Deeds Office procedures in the eighteenth century. A well designed modern system should make such lapses impossible.

The basic unit of the cadastre is the single parcel of land which is called by many names such as 'erf', 'lot', 'farm', 'block','plot' etc. It will be referred to here as a 'lot'. What follows is based loosely on the South African survey system though the principles apply to many other systems.

When viewed as an legal object, a lot comes into being when it is created, either by a grant of unalienated state land, by sub-division of an existing lot, or by the consolidation of two or more existing lots. It is important to realise that a lot may be demarcated and surveyed but it only comes into existence as a legal entity on registration i.e. when it is transferred to, and registered in the name of a new owner, or when the owner obtains a Certificate of Registered Title (CRT) in respect of that lot.

When a lot is created by Deed of Grant the land alienated is subject to the general law of property in force at the time of the grant as well as any special conditions of title which may be imposed by the grantor (the state). In principle the terms and conditions of the grant remain attached to that particular parcel of land for ever, though they may be varied by legislation or legal action for the removal of restrictions on the use of the land. When the land is sub-divided the same terms and conditions will be attached to each new lot created, and, in addition, the new lot may have further conditions of title attached. When several lots are consolidated each part of the land comprised in the new Certificate of Consolidated Title (CCT) may have different conditions of title depending on which parent lot it was included in; for this reason the boundaries of the component portions are indicated on survey plans prepared for consolidated title. It is clear therefore that in an

efficient cadastral LIS it must be possible to trace a particular area of land right back to the original grant so as to obtain all the relevant terms and conditions of title pertaining to that land. A problem which arises is that normally only graphical data showing the current state of land holdings is required.

Most of the data contained in a land information system has a temporal as well as a spatial component. When environmental data is monitored at regular intervals, different map coverages or layers can be used to store the data at each epoch. For cadastral systems this is not practical,- creating a new coverage every time a lot changed would fill available memory in a very short time. Most systems in use at present have not been designed with this factor in mind and in general facilities for tracking changes are inadequate. This ability is a key requirement for a cadastral LIS.

A cadastral system holds data about property holdings and is "lot" or "parcel" based. Every lot has a history. A lot may come into being through a grant by the State, through the recognition of existing occupation rights at the time the cadastral system was established, or as a result of sub-division or consolidation of existing lots. Lots may change their size and shape through sub-divisions being deducted, and finally lots may cease to exist when they are sub-divided without leaving a remainder or when they are consolidated with another lot or lots to form a new lot.

It is important to know not only the current status of each lot but also its history. One reason for this is that the conditions of title under which land grants are made, change from time to time as legislation changes. Such conditions of title normally apply to all sub-divisions of the original lot and so, to obtain the conditions, it is necessary to refer to the original grant.

Historic data can be maintained fairly easily as non-graphic data simply by including a field on each lot record for the "parent" lot and for the "child" lots or sub-divisions. These fields provide a link to other records which can be followed back to the original grant or forward to the current situation.

When a graphic record is kept on paper it is fairly straightforward to maintain the history. This is commonly done by using two types of plans - index sheets (also called "noting sheets", "compilations", or "standard sheets"), and lot plans ("diagrams", "certified plans", "general plans", "survey plans" etc.). The index plans show the position of each lot relative to every other lot. It is also amended so as to reflect the current position. It is not possible to keep historical data reliably on the index sheets because once a lot has been sub-divided twice it cannot be seen whether the second sub-division was cut from the first sub-division or from the parent lot. The lot plans are used to solve this problem - when a lot is sub-divided the new lot (or lots) is marked on the plan of the parent lot. When the new sub-divisions are themselves cut the new lots are marked on their own plans.

On the other hand, when graphic data is held in a computer the situation is more complicated. Most LIS systems operate with a continuous cover equivalent to the index plans of the paper system but do not maintain separate files or 'plans' of individual lots.

The method adopted by H M Land Registry in England is to transfer both graphic and attribute data relating to a parent lot which has been sub-divided to a "historical" file[27]. In

---

27 Sharman D, H M Land Registry & Ordnance Survey Digital Mapping, Land & Minerals Surveying, January 1986.

effect this implies the creation of a lot plan after a lot has been subdivided or consolidated with another lot.

A brief mention was made above (3.3.5) of current research into 'temporal databases'. Such databases are likely to provide a solution to this problem. Another way to tackle this problem is by using an abstract data type to encapsulate the land parcel, this approach is followed in Section 5.5.

4. CONCEPTS IN LIS

The second chapter dealt with the historical development of the modern computerised land or geographic information system. Systems for cadastral and for environmental data have developed from different starting points to meet different requirements. The concept of the 'multi-purpose cadastre' has been introduced in order to combine environmental data such as soil types, rainfall patterns, slope etc. with legal cadastral data to form a single data base. Rokos[1] has defined **development** as the **integration** of the possibilities of the **natural** + the **socio-economic** realities. This is precisely the information supplied by the multi-purpose cadastre. This chapter will look at some of the concepts and methods used in building land information systems.

4.1 Building a System

Virtually every country maintains some kind of land records. The start of a computerised LIS is often a decision to improve the efficiency of the existing system by introducing computer technology. According to Lyons[2], when a decision is taken to computerise, one of three broad paths may be followed:

Automating the existing (manual) system

Streamlining the existing system

Changing the existing system

---

1 Rokos D K, The Contribution of an Integrated Cadastral LIS for Decision Making, FIG XVII International Congress, Toronto, 1986, Commission 3 - Land Information Systems, pp.150-164.

2 Lyons K J, Eden E J, Chen C C, Some Thoughts on Designing Land Information Systems to Fit into a Country's Social and Administrative Framework, Regional Seminar on LIS in Pacific Rim, July 1987.

The first two paths make no fundamental changes to the organisation. Jobs may be threatened but they will be low-level clerical and drafting jobs. On the other hand, the increase in productivity will be quite modest, in the words of Humphreys 'I have visited about 800 "systems" in about 30 countries. My synopsis is that the vast majority of such systems are merely automating the inefficiencies of the past.'[3] In other words if the greatest benefits are to be derived from the new technology, the whole system must be thoroughly studied and the goals to be achieved from computerisation must be clearly specified.

4.2 System Design

An LIS is a complex system consisting of data gatherers, data managers, and data users. A central feature of the system is often a computer database management system, with graphic terminals and software for spatial data analysis, but this is **NOT** the system. LIS software can be bought 'off-the-shelf' but it must be remembered that one is not buying an LIS system. In most cases the software has to be 'customised' in order to meet the functional requirements of the system in which it is installed. Hence one may speak of 'choosing LIS software' but not of 'choosing an LIS' - a land information system has to be carefully designed for the environment in which it is to operate.

As with any information system, planning and design play a very important role. A preliminary study will look at questions such as

Who will run the system?

What data is to be kept?

---

3 Humphreys B., The Crisis in Land Management, National Seminar on Land Information Systems, Kuala Lumpur, 1984

Where does the data come from?

Who will use it?

Can revenue be generated by sale of data?

These questions must be looked at from a broad perspective because in many cases potential benefits may be much greater if a **new** system is designed rather than simply automating an existing manual system. For example, a study by the Cape Town Municiality found that the Council was maintaining no less than 25 separate database systems containing spatial data (see Appendix C). LIS planning aims to integrate all these systems using the topographic and cadastral map base.

The next step will be a **cost-benefit analysis** to see whether gains through computerisation will be sufficient to offset the cost of setting up and running the system. This is obviously a very difficult task as many of the benefits of the system cannot be foreseen or valued during the planning stage. Nevertheless in order to obtain funds it is necessary to convince those holding the purse strings that the project is commercially viable. In his paper "The Politics of Land Information Systems"[4] Hart notes that, as it is almost impossible to get funding for a complete LIS, the tendency is to divide the system into a number of sub-systems which can be justified separately, and to implement each as money becomes available. This is not entirely satisfactory because problems will arise later when the sub-systems have to be integrated.

The costs of setting up an LIS can be divided into recurrent and non-recurrent costs. Non-recurring costs are those which occur once when the system is set up and include the cost of

---

4 Hart A L,The Politics of Land Information Systems, FIG XVII International Congress, Toronto, 1986, Commission 3 - Land Information Systems, pp.267-278.

hardware, software, and of data conversion (from paper to computer records). Reasonable estimates can be made for these costs based on the experience of many organisations.

Recurrent costs are those which arise year by year to keep the system operational. Running costs include maintenance of hardware and software, salaries of data-processing staff to run the system, and costs of on-going data capture e.g. surveys for map revision.

Direct benefits will arise from reduction in clerical and drafting staff (though this may well be more than offset by the higher cost of DP staff). The major benefits though, are much harder to assess. They arise from the time saved in processing records and in searching for data, and from statistical analysis not possible under manual systems. A further unknown factor when data is sold to the public is whether faster and more convenient searches will result in an increased demand. Indirect benefits will be gained through quicker and better decisions on development projects leading to better projects.

It is likely that the benefits derived from the system are not directly proportional to the amount of data in the system but that they bear an exponential relationship. This will result in a greatly increased flow of benefits relative to expenditure in later years compared with earlier years.

According to Epstein and Duchesneau[5] cost-benefit analysis

"is often seen as an attempt to substitute mechanical methods for reasoning in administrative decision-making. Benefit-cost analysis makes no decisions, nor is it an algorithm which produces a number that solves all decision-making

---

5 Epstein E, Duchesneau T, The Use and Value of a Geodetic Reference System, Report for US Federal Geodetic Control Committee, 1984

problems...The contribution of the benefit cost approach to public expenditure analysis is that decision makers are forced to consider all factors - the quantifable and non-quantifable - in making decisions. As a result, implicit factors often become explicit. The decision process tends to focus more directly on critical elements, and orderliness and structure are imposed on the decision-making process."

4.3 The Organisation

Due to its very nature, an information system operates within an organisation. Its function is to collect data from groups inside (or outside) the organisation, to combine and analyse the data from various sources, and to supply information to other groups or individuals.

Problems of organisation seldom arise when schemes are being implemented within small organisations particularly when the sources of data are internal. However the implementation of systems which rely on input from several government departments, statutory authorities, or companies often meet serious organisational difficulties. Many conference papers stress that the single most important factor in achieving a successful implementation of a scheme is that of getting the organisation right. Hart[6] makes the point that the diversity of data and functions contained in an LIS make it difficult to identify 'special interest' or 'pressure' groups who would ensure political commitment, and yet strong political commitment would seem to be a necessary requirement for the success of a land information system.

Very often the department making the first move towards the establishment of a computerised land information system is the department responsible for the operation of the cadastre.

---

6 Hart A L,The Politics of Land Information Systems, FIG XVII International Congress, Toronto, 1986, Commission 3 - Land Information Systems, pp.267-278.

However, to expand the cadastre to a multi-purpose cadastre, the co-operation of many other departments is needed. These departments are often in different Ministries so that a commitment to the project at ministerial level is essential. In Malaysia a Deputy Minister has been appointed to oversee the development of an integrated LIS. His rank and status is such as to ensure the co-operation of civil servants in different departments. Another approach is to appoint a Director of LIS in the Prime Minister's Office where he will have sufficient authority to carry out his task.

4.4 Data Conversion

When computerising an existing information system a major part of the work is the **conversion** of existing records into computer records.

Existing spatial data records may be in digital form stored in computers, in digital form on paper (lists of co-ordinates or distances and directions), or in analogue form on paper (maps).

Existing computer data will almost certainly have to be re-formatted before it can be used in another system. This is a straightforward job, and once a program has been written to do this conversion, all the data can be converted automatically.

When digital data exists on paper it is likely to be the records of surveys used to prepare maps or the maps themselves. By going to the original measurements, errors of plotting and stretching of plans can be avoided and the full survey accuracy can be retained. In some cases it may be possible to convert printed data (e.g. a co-ordinate list) automatically by using a scanner and and optical character recognition software. (See section 4.5.3 below).

Method of digitising maps are considered below.

There is more to data conversion than the technical side of data entry. A plan for the conversion must be prepared dealing with such points as:

The methods to be used.

The order in which source documents will be processed. Normally more densely populated and economically active areas will be tackled first, because demand for this data will be greatest. It may be that revenue from the sale of this data can finance the conversion of data relating to more out of the way regions.

The completion date for the work.

The number of staff and equipment which will be needed in order to meet this completion date.

A consideration of whether it would be more effective to sub-contract the conversion job to an outside agency bearing in mind that staff required for data conversion will be redundant when the task is completed.

4.5 Input and Input Devices

'Data conversion' involves not only converting the data into a format suitable for the computer, but actually entering it into the computer. In the past there was a clear distinction between 'on-line' data entry and 'batch' processing, The widespread use of micro-computers (which may also be connected to one another and to a main-frame computer) as data-entry terminals has blurred this distinction. In fact even when data entry appears to be on-line, the data is usually sent to a 'transaction' file which is subsequently used for a batch mode updating of the main data base. This is to enable the data to be verified before any alterations are made to the database.

Various devices are used to enter data into a LIS system. Some of the commonly used ones are described below.

4.5.1 Visual Display Unit (VDU)

The most common device for communicating with a computer is the **visual display unit** (VDU). This comprises a keyboard and a display screen. The VDU has developed from 'teletype' machines. The VDU is best for interactive work - a query to the database may be typed and the result read from the screen. Some VDU's have the capacity to display graphics; these are much more useful in the context of LIS than those which can only display text. Their cost is also rather higher. A recent trend is for the cost of high-resolution graphic display units to fall sharply.

The graphic display screen is a 'raster' device. This means that the image on the screen is made up from a grid of dots or 'pixels' (picture cells). Text and other images are formed by illuminating certain groups of pixels while others remain the background colour. Each letter is therefore made up of a set of dots. Clearly the more dots there are, the closer the letter will be to a hand-drawn letter. The number of dots is called the 'resolution' of the screen. With the graphic display, the higher the resolution the easier it is for the operator to perform his task because all screen images are much sharper. Unfortunately the cost of graphic displays increases disproportionately with resolution.

The VDU is a very labour-intensive input device and this is most apparent when a large volume of paper records have to be computerised. All data entered must be typed on the keyboard. This provides ample opportunity for errors to occur. When the accuracy of the data is of critical importance, the data entry is commonly verified by re-typing, the computer detecting discrepancies between the first and second entries.

The VDU is also very inefficient for the entry of graphic data which must be typed in as lists of lines, co-ordinates, labels etc.

When the data volume is small as is the case with new transactions (i.e. the on-going updating of the database to reflect changes), the VDU provides a cheap and efficient method of entry for textual data.

4.5.1.1 Pointing Devices

For interactive graphic editing it is necessary to be able to move a graphic cursor in the screen. In this way the operator can select graphic elements for editing, draw graphic elements, postion text etc. At the most basic level the cursor can be moved with the keyboard cursor keys. Slightly more sophisticated devices for moving the cursor are the 'tracker ball' and 'joystick'. The most popular device however, is the 'mouse'. The mouse is a small hand-held object which is moved about on the table top. A ball in the base rotates according to the movements and generates electronic signals which are fed to the computer to control the cursor. Most mice have two or more control "buttons". By pressing these buttons various control characters can be sent to the computer.

4.5.2 Digitiser

As is well known, all data stored in computers is 'digital' data. Hence any data conversion which has as its object the conversion of paper records into computer records, may be regarded as the 'digitising' of the data. However in the world of interactive computer graphics 'digitising' has a special meaning - the use of an instrument called a 'digitiser', described below, to scale co-ordinates and transmit them to the computer.

Spatial data is commonly shown on maps. The map is made up of a number of graphic elements such as the 'point' 'line'

'label' and 'symbol'. Areal elements are implicit on the paper map and depicted by areas. To enter a map into a computer through a keyboard would be a very arduous task. It would be necessary to scale co-ordinates at every bend on every line on the map and then to type these values in together with the lines joining them.

The 'digitiser' is a device consisting of a flat rectangular surface or 'table' and a movable cursor. In the conventional design a mesh of fine wires is embedded in the surface of the table. Interaction between currents flowing in this mesh and the cursor enable the position of the cursor to be determined by a micro-processor in the digitiser table. This position is transmitted continuously (stream mode), or on request (point mode), to the computer. There are also one or more 'buttons' on the cursor. When one of these is pressed a code is transmitted to the computer. The codes transmitted by the 'buttons' allow the computer program to take appropriate action in response to the operators' commands.

In an alternative design an audio signal is emitted by the cursor, This signal is received by two sensors. The transit time of the signal enables the cursor position to be determined.

In effect, the digitiser scales the co-ordinates of the cursor position on its own (physical) grid. Should a button be pressed at this point the computer is able to identify a specific point on the map with a particular digitiser co-ordinate.

From a physical point of view the conventional digitiser is a 'raster' device because it is dependant on a grid of wires. Its resolution depends on how close together these wires are, and its accuracy on how accurately they are placed in the digitiser table. From the 'logical' viewpoint the digitiser is a 'vector' device, the graphic objects it captures are

points which are transmitted to the computer as co-ordinates. The digitiser co-ordinates have to be converted to 'world' or map co-ordinates before they can be used. Transformation software performs this task.

The digitiser is also often used as a control device for the computer. A menu of commands is placed on the digitising tablet. When a point within the menu is selected, the computer determines which command has been selected and takes the appropriate action.

4.5.3 Scanner

The scanner is a 'raster' device which captures data in 'raster' form. A beam of light is passed from side to side over the document to be scanned. It covers the document by scanning a series of rows. This can be done either by moving the scanning beam down the paper or by moving the paper in front of the beam. A detector records the light reflected from the scanning beam by the map. The reflected light will have maximum intensity where the map is white and minimum where it is black (When the map to be scanned is transparent, a white backing must be placed behind it). Other colours, or shades of grey, will give intensities between white and black. Simple scanners have a cut-off point and regard points to one side of it as white and to the other side as black. More sophisticated devices assign a 'grey scale' to scanned points based on the intensity of the light reflected from them. Colour scanners do exist but are much more expensive than monochrome ones because separate detectors and filters are needed for each primary colour. Scanners typically have a resolution of about 300 dots per inch.

Systems which require high accuracy (e.g. to maintain survey accuracy) without an excessively large data storage requirement, must be vector based i.e. objects are stored as points, lines, and polygons. The output from the scanner however is a set of raster or grid data. Before this data can

be added to a vector-based system the lines in the raster data must be identified and expressed as vectors. Once the lines have been converted to vectors standard algorithms may be applied to form polygons.

In principle vectorisation is quite straightforward as shown in the figure below. In practice however many problems are encountered due to irregularities in the source document such as variations in the width of lines, small breaks in lines, and dirt marks. With more sophisticated algorithms and greater processing power these problems are to a large extent being overcome[7],[8].

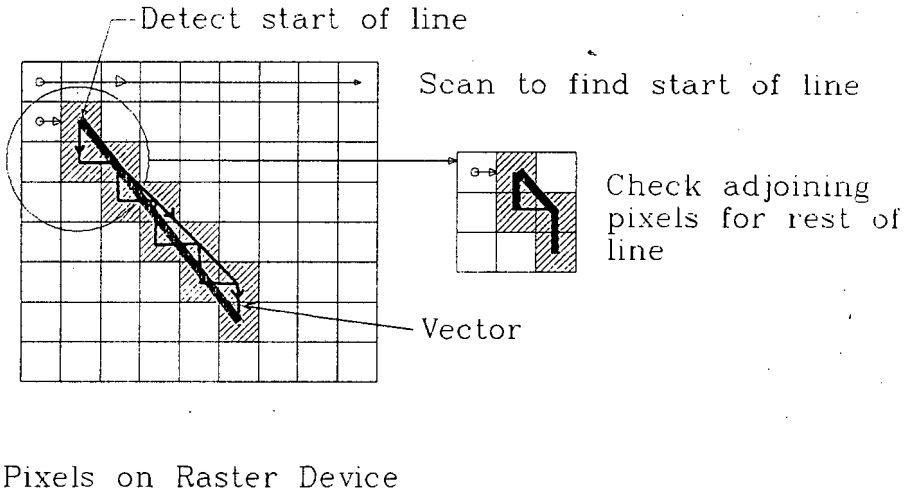


Fig. 4 - 1 Vectorisation

A second problem that arises with scanning is the categorisation of map features. When maps are digitised, the operator enters a code to indicate the feature being digitised, whether it is a road, a fence, a school or a house. The raster file contains no such information and it must be added by manual editing - a time-consuming process.

---

7 Dale P, De Simone M, Vector-Raster-Vector Conversions, Land & Minerals Surveying, March 1986.

8 Hodges D J, Cooper S M, Computer-Aided Digitisation of Maps and Plans, Land & Mineral Surveying, June 1988.

Efforts have been made to develop automatic systems for identifying and coding features[9]. Such systems rely on artificial intelligence methods. The program is controlled by a set of rules, and feature codes are assigned to objects according to the rules. For example a set of parallel lines 1m apart would be interpreted as a railway, while if they were 10m apart they would be interpreted as a road. Similarly a small rectangular object would be a building and a large one a playing field. Use can also be made of line thickness, for example cadastral boundaries may always be drawn 0.05 mm thick, the computer will then interpret all lines of this thickness as cadastral boundaries. Unfortunately with hand-drawn originals this is not very reliable.

Tests at North East London Polytechnic[10] have shown that up to 90% accurate automatic feature coding is possible in some circumstances. Nevertheless the cost of checking and editing leads the majority of people on the production side to believe that digitising ab initio is a better way to bring data into an LIS system.

Pattern recognition software is being developed primarily in the field of 'machine vision' for the control of industrial robots. Such software matches raster images, taken from scanner or camera, with images stored in the computer memory. Artifical intelligence techniques are used to reduce the seach time and to obtain the best match when no exact match is found. This type of software holds promise for faster and more reliable automatic feature coding and map interpretation.

---

9 Dale P, De Simone M, The Automatic Recognition of Features in Digital Maps, Land & Minerals Surveying, June 1986.

10 ibid.

The scanner can also read printed or typewritten documents. Optical character recognition (OCR) software (a sub-set of pattern recognition software) is able to convert the raster scan into a text file. Dirt or blemishes on the paper and non-alphabetic characters confuse this software which works best on a clean white page. When the original documents are of good quality this provides an economic alternative to typing all the text into the computer. At present OCR software cannot handle hand-written documents reliably.

4.5.4 Line Scanners

A recent development is 'interactive scanning'. This is similar to digitising but the operator does not follow every object to be digitised with a cursor. Instead he places a scanner on that feature - the scanner then automatically follows the feature. If the machine cannot resolve any problem e.g. a break in the line, overshoot, nodal point, it waits for operator action to resolve the problem. This system is said to be faster and more accurate than conventional digitising while maintaining the advantages such as the ability to enter feature codes for the data elements being digitised.

4.5.5 A Comparison of Digitising and Scanning[11,12]

The vector representation of maps is generally more useful in the LIS context because

survey accuracy can be preserved

ground features form 'objects' to which attribute data can easily be attached

---

11 Woodsford P, Data Acquisition: the Case for Vector, Land & Mineral Surveying, January 1988.

12 Dirdal P, Data Acquisition: the Case for Raster, Land & Mineral Surveying, January 1988.

data files are substantially smaller than is the case with raster files.

The chief drawback to the vector representation is the time taken and the high cost involved in digitising (vectorising) maps. After 15 years of digitising, the Ordnance Survey in UK has still only digitised 10% of its stock of maps.

Scanning presents a quick solution to the problem of getting maps into a digital (raster) form. Raster data files though, are very much larger than those for vector data. The accuracy of the data is related to the accuracy of the source document whether the data is captured by scanning or by digitising, however the vector system allows the possibility of capturing the data from the original survey observations.

Raster data is particularly useful for providing a background map on which vector data is displayed. Such raster maps can be kept on video-discs (CD-ROM).

At present the line-scanner promises to overcome the disadvantages of both digitiser and scanner.

4.5.6 Camera

The digital camera is a raster device similar in principle to the scanner. In the scanner a light-sensitive device is moved over a document to 'scan' it. The digital camera has an array of light sensitive devices in place of the film of a normal camera. An image is brought to a focus on this array and the pattern of light and shade creates varying currents in the array. The currents in the various devices making up the array or grid are read sequentially by the computer to produce a file similar to that produced by a scanner. Remarks above about data interpretation when using a scanner apply equally to camera images.

The advantage that the camera has over the scanner is that it is much more versatile. Digital images captured by cameras in

aircraft and satellites can be used for automatic map and terrain model production. This topic though, is beyond the scope of this thesis.

The big disadvantage of the camera for the conversion of existing maps and plans is its low resolution. A typical camera will break a scene up into 600 x 600 pixels. Very high resolution cameras may reach 3000 x 3000 pixels. Using a camera with a resolution of 2000 x 2000 pixels, a 7" x 7" map may be captured with the same pixel size as a 300 dot per sq. in. scanner. For a plan of A1 size the resolution will be of the order of 75 pixels per inch. For this reason the use of digital cameras for data conversion is very limited.

4.6 Data Collection

The transfer of existing data from storage on paper, micro-film etc. to a computer database has been discussed above. However land information systems are not static. Planning schemes are continuously amended to meet changing circumstances, land is sub-divided and consolidated, buildings are erected and demolished, environment changes occur as a result of industrial pollution, bush fires, floods, drought etc. Therefore an LIS must be continually updated if it is to retain its value.

Data collection takes place in several ways - attribute data relating to spatial data, which has already been mapped, is gathered by inspection, observation, and questionnaires. Land use, demographic, and environmental data is best collected either by noting the results of inspection on a map, or by connecting the answers to questionnaires to street addresses. The importance of street addresses as a key to spatial databases was noted in 3.1 above.

Source Data Automation is a term used in the data processing environment which means the capture of data at source by electronic means. A typical example is the use of a data

recorder or "electronic field book" to record survey observations directly from the "total station" (A total station is an instrument combining an electronic theodolite and distance measurer. It measures angles and distances and presents the data in a digital form). This procedure eliminates errors made by reading the instrument incorrectly, or by writing down incorrectly what was read. The labour of manual transcription of data from the field book to the computer is also eliminated, as are errors made at this stage of the work. For land use surveys, a small data recorder such as the "Psion Organiser" can be used, the surveyor simply entering the erf number and a code representing the particular use.

Specialised recording instruments are available for certain types of data collection e.g. automatic tide guages and meteorological stations. Data from such instruments can be transfered directly to the LIS computer.

For at least sixty years aerial photogrammetry has been a very important way for gathering spatial data. In recent years the types of data which can be collected has been expanded by the use of multi-spectral scanners (MSS) in aircraft and satellites. This type of equipment produces spatial data in 'raster format'. Essentially the position of the picture is tied to a fixed spatial location by identifying objects in the picture which have a known position. Although a great deal of attribute data can be obtained from photographs or scanned images, in most cases field verification and annotation remains essential. The problem of vectorising raster data was mentioned above, but it is not always necessary to do this. For some purposes e.g. environmental or demographic, it is possible to keep the data in raster form. Research has been taking place to develop a link between a standard relational database and a set of raster images.

4.7 Output

Spatial data is conventionally depicted on maps. A paper map is in itself a land information system providing information on the position of towns, rivers, mountains, schools, police posts, roads, airfields etc. The map has been found to be the best way of showing such data, and people requiring this information are familiar with the use of maps. The computer system must therefore be able to produce maps.

Apart from spatial data, every information system will also contain textual data which is linked to the spatial elements. Such data is commonly referred to as "attribute" or "aspatial" data because it supplies the attributes of the spatial elements. This data is most often kept and used in text form. Producing data listings in text form presents no difficulties. For example some lots (or erven) may be selected on a graphic display - data such as owner's name, address, purchase price, land use could be printed as a text listing.

Text listings are not always the best way to present attribute data. When data can be aggregated and analysed, thematic maps, graphs, bar charts and pie charts become useful tools for the display of information. The message contained in data can be brought out more clearly through displaying it in the most appropriate way.

4.7.1 Maps

The first steps in computerised map production were taken almost thirty years ago. At that time graphic displays were primitive and expensive but both drum and flat-bed plotters were beginning to appear on the market. Because of cost the use of this type of equipment was limited initially to large companies and government departments. By the late sixties private surveyors were beginning to make use of plotters for automatic plan drawing. However, because the plan could not

be previewed on a graphic terminal, because editing was extremely difficult, and because the cost of plotters did not fall as fast as that of other computer peripherals, the use of plotters did not spread very fast.

In the early eighties the cost of interactive graphic display terminals fell to the point where there was a mass market. This has lead to a vast output of software for computer aided drafting and design (CAD). This technology, both hardware and software, has been applied to mapping and is the reason for the enormous increase in popularity of land information systems. The map can now be viewed and edited on the screen before being sent to the plotter. This means that a final product can be produced without the need for touching up by a draftsman.

4.7.2 Digital Terrain Modelling

For some purposes two dimensional representation of three dimensional data is not adequate. These purposes include road design, where three dimensional data is needed to compute slopes and cut and fill volumes, and military use, to enable missiles and aircraft to fly very close to the ground in all conditions. To meet these requirements the **'digital terrain model'** is created. This is not strictly speaking a three dimensional model of the ground but just of the surface of the ground (i.e. underground features are not shown). Perspective views or projections of the surface can be generated from any position, and these can be displayed on a graphic terminal or plotted on paper. On the graphic terminal the surface or **'DTM'** can be rotated, tilted, and viewed from any angle. This makes it much easier for a planner or architect to visualise his design.

The term **'Digitial Elevation Model'** or **'DEM'** is also used. This usually has a more limited meaning, refering to an array of spot heights or a set of contours which contain no additional information about the terrain.

Most computer-aided mapping systems (which always form part of an LIS) are able to create contours from spot heights, and can create DTM's from either contours or from spot heights. At present the DTM is not seen to be part of the spatial data, but rather as the answer to a query "What does the surface look like?".

The term 'geomatic database' has been coined for a database which contains altimetry and planimetric data. A number of systems have been developed to give a dynamic view of the landscape view from a moving position, amongst which is one called 'GeoGraph'[13]. This type of display is different from the more simple DTM in that the terrain model is shown with topographic features such as buildings, roads, rivers, forests etc superimposed on it. It is in fact, a simulation of the landscape. Such a database requires data such as the heights of buildings, pylons, trees etc which have generally not been captured during mapping operations. Surface simulations of this kind are very useful for planning as the planner can put his design into the computer, and then view its effect on the landscape from every angle. They are also very useful in training pilots for low-level flying.

4.7.3 Other Output - Data Exchange

Apart from output to paper and display screen, computers can readily supply data in digital form, either directly to other computers through communication links, or indirectly through storage media such as magnetic and optical disks and tapes. Magnetic tapes have been used as a medium for the exchange of textual data for many years and there are well established standards. In the case of graphic data, exchange is more difficult as standards are not yet well established and the

---

13 Couesnou T, Laurent D, Motet S, The Geo-Graph Simulation System: Towards Dynamic Use of a Geomatic Data Base, Advanced Computer Graphics: Proceedings of Computer Graphics in Tokyo '86, Springer-Verlag, Tokyo 1986.

data is much more complex - all computer systems can reproduce text (in the Roman alphabet), but not all systems have facilities for patterning polygons or drawing ellipses.

An effective standard for the exchange with other systems of the data, both graphic and non-graphic, stored within an LIS is an essential requirement for the future. This is because the situation either already exists or soon will, that different authorities needing to share data are using different computer-aided mapping or LIS systems. e.g. in Singapore the Survey Department and the Public Utilities Board use Syscan systems, the Housing Board uses ComputerVision; Port of Singapore Authority, Jurong Town Corporation, and Telecoms use Intergraph, and the Planning Department a Sicad system. There is clearly far less work involved in developing translators for each system which will translate the internal data format to or from a universal standard rather than direct translators between each system involved. If the latter approach was followed in the example given above, six translator modules would be required rather than four if a universal standard were adopted. If another type of system were to be introduced to the environment, four new translator modules would be required in the first case but only one in the latter.

Some current standards for the exchange of graphic data are given below:-

IGES - One of the first general data exchange formats was the Initial Graphic Exchange Format (IGES) developed by the U.S. National Bureau of Standards. This is described thus "This Specification (format) is concerned with the data required to describe and communicate the essential engineering characteristics of physical objects as manufactured-products. Such products are described in terms of their physical shape, their dimensions, and information which further describes or explains the product... The requirements for a common data

communication format for product definition can be understood in terms of today's CAD/CAM environment... which should be able to exchange product definition data, but which usually employs incompatible data representation and formats. In addressing this compatibility this Specification is concerned with needs and capabilities of current and advanced methods of CAD/CAM product definition."[14] From this it can be seen that IGES is aiming at a very broad capability, nevertheless it has been, and is, used for the transfer of digital cartographic data. It is provided with AUTOCAD and Intergraph systems to name but two. It is held by Nyerges[15] that the IGES standard is not appropriate for cartographic data exchange because it lacks spatial referencing system information, data quality information, and raster and grid cell information. The first is not a valid objection when all systems are referred to a common national control system; in many cases the second objection will be irrelevant because few cartographic systems have a reliability code attached to each graphic element.

SIF - The Standard Interchange Format (SIF) was introduced by the Intergraph Corporation in 1986. It is defined as "a generic format for transmittal of graphic data and associated non-graphic data between systems"[16]. SIF has been recommended

---

14 U.S. National Bureau of Standards, Initial Graphic Exchange Specification (IGES) Version 2.0, U.S. Department of Commerce, National Technical Information Centre, Publication No.PB83-127448, Washington D.C., 1983.

15 Nyerges T, Testing the Interim Proposed Standard for Cartographic Data Exchange, Digital Cartographic Data Standards: A Report on Evaluation and Empirical Testing, Report No.7, National Committee for Digital Cartographic Data Standards (NCDCDS), 1986.

16 Intergraph Corporation, Standard Interchange Format (SIF) Command Language Implementation Guide, Intergraph Corporation Document No. DIXD7330, Huntsville, Alabama, 1986.

as the standard to be used in Singapore's proposed 'Land Data Sharing Project'. One reason for this is undoubtedly that more Intergraph systems are in use than any other and these are already supplied with a SIF translator.

FGEF - Federal Geographic Exchange Format. This format has been developed by the Federal Interagency Co-ordinating Committee on Digital Cartography to "facilitate the exchange of spatial data among Federal Agencies ... The format provides a means for exchanging geographic point, line, polygon and gridded data and associated attributes, as well as features which can be a combination of the four types."[17] This is not a general purpose spatial data exchange format but has been developed specifically for cartographic data.

Data exchange formats have also been developed in Australia[18], the U.K.[19], and South Africa[20]. Due to the different data structures used by different software systems, it has proved very difficult to transfer topology. The standard procedure is to transfer lines and points; the importing software then re-creates the topology.

Data exchange formats only cover the transfer of graphic and textual data between two systems, they do not ensure that the meaning is the same to the organisation which receives the data as it was to the organisation which sent it. For example, feature code '1001' means a first class road to A.

---

17 Federal Geographic Exchange Format, Federal Interagency Co-ordinating Committee on Digital Cartography, 1986.

18 Interchange of Feature Coded Digital Mapping Data: Australian Standard 2482-1981, Standards Association of Australia, 1981.

19 The National Transfer Format Release 1.0, Ordnance Survey, Southampton, 1987.

20 A National Standard for the Exchange of Digital Geo-referenced Information, National Research Institute for Mathematical Sciences, Pretoria, 1987.

He transfers some map data to B. The first class roads are transfered correctly according to the data exchange protocol - B receives a digital map sheet with a set of lines with feature code '1001' attached to them. Unfortunately in B's system code '1001' means a footpath. Hence a protocol for data exchange is needed and also a translation table for feature codes. The South African National Standard for the Exchange of Digital Geo-Referenced Information recognises this problem and offers a Standard Feature Classification in Appendix B. The document notes that:

"It is imperative that the standard feature classification scheme togther with the feature coding scheme and list of non-spatial attributes be maintained centrally by a national co-ordinating body."

Land use is another area which lends itself to classification. Various national land use codes exist e.g. The South African National Land Use Code (1984).

4.8 Output devices

In some cases, as has been mentioned above, the end user of the data will want it in digital form to use on his own computer system. However, in most cases the users still need data on paper. A brief description of some of the output devices used in the LIS environment follows:-

4.8.1 Printers

The printer plays a very important role in any information system because it is the means by which the user is able to get a permanent copy of the non-graphic or attribute data in a form readable by humans.

There is an enormous range in the speed, print-quality, and cost of printers. Printers may be classified as either 'impact' or 'non-impact' type.

'Impact' printers print on the same principle as a typewriter. A ribbon impregnated with ink is pressed against the paper so as to form the outline of the desired character. For this reason only 'impact' printers can be used with carbon paper to produce multiple copies. Printers in this category range from 'daisywheel' and dot-matrix printers with printing speeds from 10 to 200 characters per second to 'drum' printers capable of 2500 charcters per second or about 30 pages per minute.

The dot-matrix printer has a print-head with (commonly) either 9 or 24 pins. The print-head moves backwards and forwards across the paper on a carriage. The pins are fired against the paper under software control so as to form text. Because the text generation is software controlled it is a simple matter to print in non-Roman alphbets such as Arabic or Cyrillic. The dot-matrix printer can also produce graphics, though the resolution is usually limited to about 60 dots per centimetre. Because dot-matrix printers are versatile - they can print text and graphics, and beacuse they are cheap (US$ 150 - $1000), one is to be found next to almost every computer.

If large volumes of output are required a 'drum' or 'line' printer will be chosen. The term 'drum' arises because a drum rotates continuously behind the paper. The drum has a complete character set embossed on it for each character position on the paper. When the appropriate character on the drum passes the paper, a hammer strikes the character against the paper. In this way a complete line of text is printed during a single rotation of the drum - hence the name 'line' printer, since a complete line is printed at one time. Line printers cost considerably more than dot-matrix printers and cannot handle graphics, but because of their speed, they are the choice when large print volumes with multiple copies are needed.

Many types of non-impact printers have been produced using such techniques as a hot wire or laser beam on heat-sensitive paper, 'ink-jet' or liquid ink squirted through a nozzle at the paper, laser beam on photographic material, or electro-static methods. Some of this equipment has found specialist market niches e.g. the laser-photographic printer where very high speed is needed (up to 6 pages per second), or the thermal paper printer where silence of operation is important. However the only non-impact printer for which a mass market has developed is the 'laser' printer which operates on the same electro-static principle used in 'plain paper' photo-copying machines. The laser printer typically costs from US$ 1000 to $ 5000, can print eight pages per minute, can use many print fonts, and can reproduce graphics at a resolution of 60 dots per centimetre. The excellent print quality means that the 'laser' printer will displace the dot-matrix as the printer on every table if prices continue to fall as they have done over the past five years.

The electro-static plotter which is described below is essentially a large 'laser' printer.

Colour printers, both 'impact' (using multi-colour ribbons) and 'non-impact' (ink-jet) have been on the market for several years. However they have not achieved great popularity because at present they are expensive to buy and operate, not very reliable, very slow in operation, and do not give a high quality result. No doubt this situation will change in the years ahead.

4.8.2 Plotters

Plotters provide a means of producing a 'hard copy' on paper or drafting film of the graphical data held in the system. Several types of plotter are in use amongst which are:

4.8.2.1 The Drum plotter

The first drum plotters consisted of a drum to which the paper was fixed and which could be rotated by a stepping motor. Another stepping motor moved a pen at right angles to the direction of the motion of the drum. In the modern version the drum is stationery and the paper is moved over the drum by rollers which grip the paper. By moving the motors the appropriate number of steps, the pen can be brought to any point on the plotting surface. A further control raises and lowers the pen. In early plotters the programmer had to convert the positions of points and lines to be plotted into actual motion of the motors. Modern plotters have built-in micro-processors which respond to a high-level command language. HPGL (Hewlett-Packard Graphic Language) has become a de facto standard.

Plotters are now available which can be programmed to select pens from a 'carousel' so that different line widths and/or different colours can be used for drawing without intervention by the operator. The pen speed is also programmable to cater for different types of pens; the speed at which ink can reach the paper is the major factor limiting the speed at which drum plotters can operate. Most drum plotters are either 30" or 40" in width and have the advantage that there is no physical constraint on the length of the paper (though there may be software constraints).

This type of plotter is regarded as a low-cost device with good performance. (US$ 5000+ for a 4 pen 40" plotter) The accuracy of a new instrument exceeds that of a human draftsman, but wear and tear on the rollers may affect accuracy.

4.8.2.2 The Flat-bed Plotter

The flat-bed plotter is essentially similar to the drum plotter. In this case the paper remains fixed to a flat

plotting table and stepping motors are used to move the pen in two directions. The flat-bed plotter is found at both ends of the price spectrum - there are very cheap plotters in sizes from A4 to A0, and very expensive plotters with large plotting tables. The expensive models are extremely accurate.

4.8.2.3 The Electro-static Plotter

The pen plotter is a vector device - the pen movements are all vector operations. The pen makes each move for a specified distance in a specified direction. The electro-static plotter, on the other hand, is a raster device and works on the same principle as the photo-copier. All data to be plotted is converted from vector to raster form before plotting. Large-format electro-static plotters are expensive and do not provide the same quality output as pen plotters but are fast and are often used for draft plots in large organisations.

The 'laser' printer also operates on the electro-static principle. Many laser printers have HPGL interfaces and thus can be used as plotters, though generally only A4 size can be plotted.

4.8.3 Display Terminals

Display terminals were dealt with in Section 4.5.1 above under 'Input Devices'. Being an interactive device, the display terminal also plays an important role in output. High resolution colour displays are useful in displaying maps for interactive editing, aiding in map composition (in enabling the final product to be viewed before it is plotted), and in answering database queries in cases when either a graphic input or output is needed.

5. SOME LIS SOFTWARE

5.1 Data Structures and Basic Manipulation

In section 3.1 above, a general outline was given of two dimensional spatial data and the methods (vector and raster) used to store this data. This section will consider vector data and develop data structures in Modula-2 to represent this data.

5.1.1 Spatial Data in Three Dimensions

The world we live in is three dimensional (and for many purposes time can be regarded as a fourth dimension), therefore it would be logical to adopt a structure for spatial data which reflected its three dimensional nature. This is not often done for several reasons. Firstly, over a city or country, differences in elevation are small in comparison with the extent in latitude and longitude. Secondly, centuries of culture have conditioned us to accept the projection of points of different elevations on to a flat map, and thirdly, the computational effort required to handle three dimensional data is much greater than with two dimensional data. Despite this, elevation is important, therefore computerised mapping systems generally follow the conventions adopted for paper maps i.e. to store contour lines and spot heights as features on a two dimensional map.

5.1.2 Spatial Data in Two Dimensions

5.1.2.1 Topological Structure

The term 'topological' is widely used in connection with spatial data to mean data structured in such a way that the relationships between polygons and lines are held within the structure. e.g. by recalling polygon 'X' from the database a program can determine that polygon 'X' is abutted by polygon 'Y' on side 1, by polygon 'Z' on side 2, etc. The benefit

gained from topological structuring is a great reduction in search times, and hence, a more efficient database.

5.1.2.2 Data Structures

In the development of the software contained in Appendix A, certain data structures have been adopted. These structures and the reasons for their choice are explained below. The purpose of this software is to illustrate some of the concepts outlined in this thesis. The data structure adopted is obviously not the only possible one, but it has the merit of functionallity and simplicity. It is similar to the structure used in ARC-INFO, a very popular commercial LIS software package.

The basic spatial element is the **point** which has position. Using two-dimensional rectangular Cartesian co-ordinates its position can be given by an X and a Y co-ordinate which represent the distances to the north of an east-west line through the origin, and to the east of a north-south line through the origin. It is clear that the co-ordinates of a points by themselves do not specify the position of the point, but only when they are taken in conjunction with the system parameters i.e. the position of the origin, the azimuth of the axes, and the units of measurement used. In addition to co-ordinates a point needs a name by which to identify it. A suitable data structure in Modula-2 would be:-

```
TYPE
     coord = RECORD
                id : CARDINAL;
                X  : LONGREAL;
                Y  : LONGREAL;
             END;
```

In a practical system, other fields would be added for such items as description of point, field-book reference, previous values etc.

The **line** is an element which connects two points and can be represented by two points. It is **not** necessary for the line data element to hold the co-ordinates of the two end points. If the line element holds only the names of the points the co-ordinate values can be obtained by reference to the point data elements. This has two advantages, firstly duplication of data is avoided, and secondly, if the co-ordinates of a point change, the line starting from that point will automatically move.

A line data type can be defined as

```
TYPE
     line = RECORD
               pt1 : CARDINAL; (* point id's *)
               pt2 : CARDINAL;
            END;
```

The use of these data types is shown in the following example:-

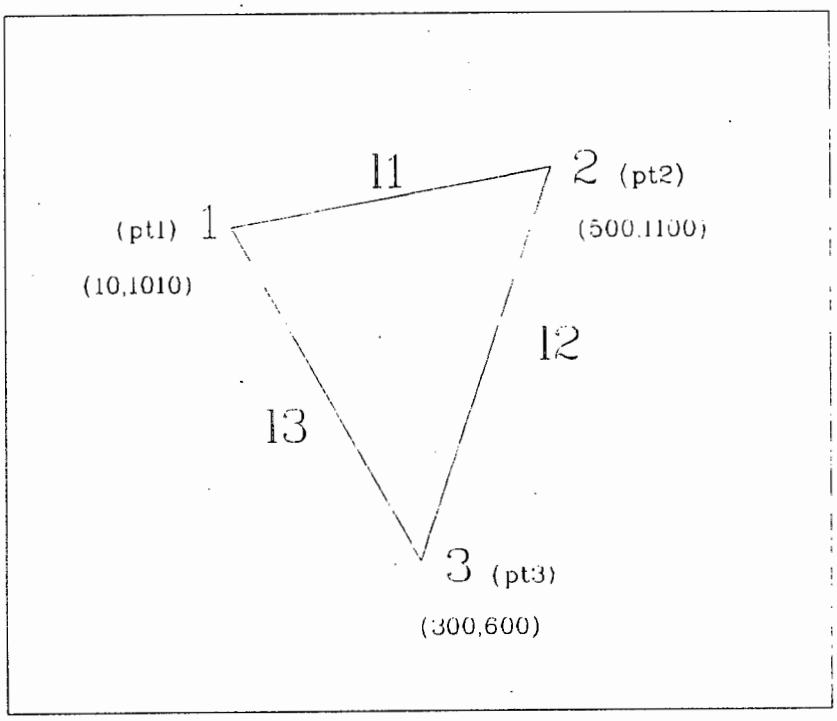


Fig. 5 - 1 Points and Lines

p1, p2, p3 are points which are represented as

p1.id = 1	p2.id = 2	p3.id = 3
p1.X = 1010	p2.X = 1100	p3.X = 600
p1.Y = 10	p2.Y = 500	p3.Y = 300

l1, l2, l3 are lines which are represented as

l1.pt1 = 1	l2.pt1 = 2	l3.pt1 = 3
l1.pt2 = 2	l2.pt2 = 3	l3.pt2 = 1

Part of the topological data is information about which lines join to which others. This is implicit in a set of lines because any lines which share a common point must join. However the information can be made more readily accessible by aggregating groups of connected lines into **'chains'** of lines (called 'arcs' in ARC-INFO, hence the name).

e.g. in the example above the lines l1, l2, l3 could be connected to form chain c1 = l1 l2 l3.

A problem which arises is how to define the route of a chain and where it starts and ends. In any network of connected lines, there will be many routes through the network but only some of these are useful for defining topological structure. This problem can be solved by classifying some points as **'nodes'**. Nodes are points where more than two lines meet. This is illustrated in Figure 5-2.

Points 1, 3, 4, 5, 7, and 8 are nodes because at least three lines meet at these points. Points 2, 6, and 9 are not nodes because only two lines meet at these points.

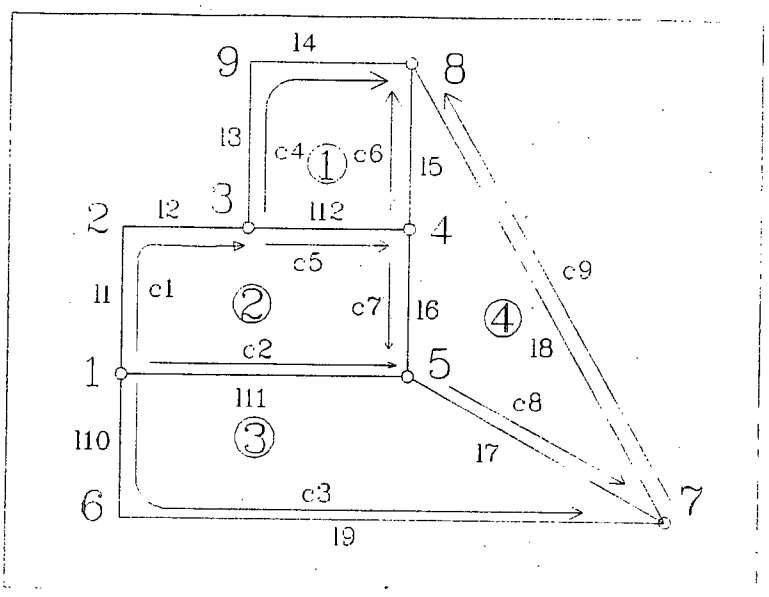


Fig. 5 - 2 A Topological Network

The **chain** may now be described as an ordered list of lines connecting two nodes. A chain may consist of one line or of many lines, hence a dynamic data type is needed to store it. (It would be very wasteful of memory to allocate say 50 lines to every chain when the average chain contained only one line, furthermore, the odd chain with 51 lines would 'crash' the system).

The starting and ending points of a chain may be found by accessing the first and last elements of the list of points defining the chain. However, to avoid some of the overheads of list access, the start and end of the list can be included as separate data elements.

The **chain** may be defined as

```
TYPE
     chain = RECORD
                  id     : CARDINAL;  name of chain
                  start  : CARDINAL;  starting point
                  end    : CARDINAL;  end point
                  left   : CARDINAL;  adjoining polygons
```

```
        right  : CARDINAL;  see below
        points : idlist:    list of point id's
    END;
```

The final data type is the **'polygon'**. This is a closed figure bounded by an arbitrary number of lines (more than two). It can therefore be represented by a connected sequence or list of chains which start and end on the same point.

In the example above polygon 1 is bounded by the lines l3, l4, l5, l12 or the chains c4, c6, c5. The chains c5 and c6 are going in the reverse direction i.e. from points 3 to 4 and 4 to 8 rather than from points 4 to 3 and 8 to 4 as in polygon 1. It is not actually necessary to distinguish this case because the end point of the previous chain and the starting point of the next chain will show in which direction the chain should be traversed. However, a convention has been adopted here of giving the chain id a negative sign if the chain has to be traversed backwards from the end to the beginning, thus polygon 1 would be c4, -c6, -c5.

A polygon might be defined as

```
TYPE
    polygon = RECORD
                  id    : CARDINAL; polygon's name
                  sides: idlist; list of chain id's
              END;
```

The boundary between any two polygons is marked by a chain. Hence every chain will have one polygon to its left, when viewed from the start, and one to its right. The particular polygon numbers are included in the chain data structure (see above) in the fields 'left' and 'right'. These fields provide a direct link from one polygon to all adjoining polygons.

Refer again to the figure 5 - 2

Chains are:-

c1	id	1	c2	id	2	c3	id	3
	start	1		start	1		start	1
	end	3		end	5		end	7
	left	-1		left	2		left	3
	right	2		right	3		right	-1
	points	1,2,3		points	1,5		points	1,6,7

c4	id	4	c5	id	5	c6	id	6
	start	3		start	3		start	4
	end	8		end	4		end	8
	left	-1		left	1		left	1
	right	1		right	2		right	4
	points	3,9,8		points	3,4		points	4,8

c7	id	7	c8	id	8	c9	id	9
	start	4		start	5		start	7
	end	5		end	7		end	8
	left	4		left	4		left	4
	right	2		right	3		right	-1
	points	4,5		points	5,7		points	7,8

Polygon -1 is the 'world' polygon i.e. anything not included in any other polygon.

The polygons will be:-

p1	id	1	p2	id	2	p3	id	3
	sides	4,-6,-5		sides	1,5,7,-2		sides	2,8,-3

p4	id	4
	sides	8,9,-6,7

Sides are chain numbers.

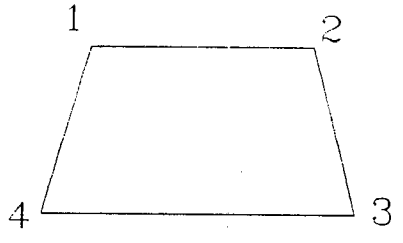


Fig. 5 - 3 Simple Polygon

There are unfortunately certain cases which the above schema cannot handle.

In this case there are no nodes as defined above, hence no chain and no polygon. This case is dealt with by adopting any point of the figure as a **'pseudonode'**. The structure can then be built up from that point.

Another common occurrence is 'holes' or 'islands' within polygons

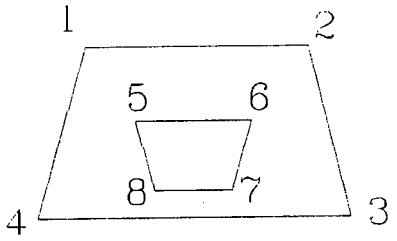


Fig. 5 - 4 A "Hole"

Area of polygon 1 is area 1-2-3-4-1 minus the area of polygon 2.

A solution to this problem is to add a field to the polygon data structure which will contain a list of polygons within the polygon. The structure is amended as follows:-

```
TYPE
     polygon = RECORD
                    id     : CARDINAL; polygon number
                    sides  : idlist;   list of chains
                    inside: idlist;    list if polygons
               END;
```

Relationships exist between the basic elements introduced above which make up two-dimensional topography - the point,

line, and polygon. The diagram below illustrates this. The relationships can be normalised to produce the tables shown in the second diagram.

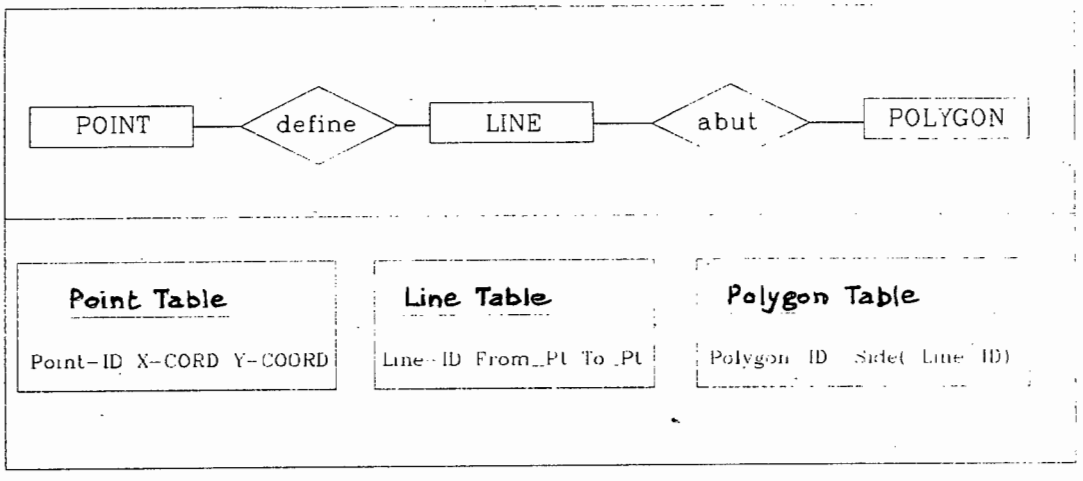


Fig. 5 - 5 Entity Relationship Diagram

In order to identify a polygon on the display terminal it must be labelled. While labels cannot strictly be regarded as forming part of the topological data, it is convenient to include them in the data structure. A label is a text string together with the co-ordinates of an insertion point. It can be included in the polygon data structure be amending the structure as follows:-

```
TYPE
     polygon = RECORD
                    id     : CARDINAL; polygon number
                    sides  : idlist;   list of chains
                    inside: idlist;    list if polygons
                    label  : ARRAY OF CHAR;
                    labx   : LONGREAL; insertion point
                    laby   : LONGREAL;
               END;
```

It was stated above (4.5.2) that lines rather than chains form the input data. It is a simple operation to convert a

set of lines to a set of chains and vice versa, therefore it is only necessary to store the chains which are referenced by the polygons and which contain adjacency data relating to the polygons. The line data as such need not be saved unless it is necessary to keep attribute data for individual lines e.g. observations, field-book references etc.

A map level, layer, or coverage will comprise a set of points, lines, chains, and polygons. A cover can be defined as follows:-

```
TYPE
     cover = RECORD
                cs : coordset;
                ls : lineset;
                ch : chainlist;
                ps : polygonset;
             END;
```

A data element of type 'cover' will encompass all the spatial data contained within a single map layer. It is important that the different data elements (point, chain, polygon) be combined into a single data type because the polygon depends on chains which in turn depend on points.

5.2 Indexing Spatial Data

When selecting or 'picking' a graphic object on the display screen it is essential that the object should be found quickly in the graphic database. If the database is large and the cursor position has to be tested against every object it might take minutes or hours to locate the object selected. For this reason the data must be indexed in some way.

Points are relatively easy to index because they only occupy one position. Lines and polygons present more of a problem. Lines occupy a continuous extent between the end points and a point in the middle of the line may be 'picked' to select the

line. The two end points might be far from the point selected and any index based on point positions would be quite useless. Some suggestions are given below in the section on indexing.

Two common indexing techniques are used:-

5.2.1 Indexing of X or Y values

If the purpose of the index is to enable a point selected on the screen to be located in the database, a simple , well-known, B-tree index on either the X or the Y co-ordinate values will serve the purpose. Using the cursor X value, the index will provide all the points with values within the tolerance for the 'pick' operation. These points can then be tested in turn to find which is the closest to the cursor. In a slightly more sophisticated form, separate B-tree indexes are built on the X and Y co-orinate values. By accessing each of these indexes in turn, two sets of points are formed - those within the tolerance on the X axis, and those within the tolerance on the Y axis. The point common to both sets is the required point. If more than one point is common they can be tested to find which is the closest.

This indexing technique can also be used to extract all the points within a given area such as a screen window, but an exhaustive search would still have to be done to locate lines starting and ending outside the area but passing through it.

When a sub-set of the data is selected for display in a screen window, a temporary file is normally opened to hold the graphic elements on the screen. Thus the search area is limited to the visible elements in the display file.

5.2.2 Quad-trees

The term 'quad-tree' is a general term used to cover any two dimensional tree structure. One form, equivalent to a two dimensional binary tree, can be used to compress raster data

files - the raster area is repeatedly subdivided into squares or quadrants until each 'leaf' contains only 'set' pixels (i.e. those pixels with a non-zero value). In some circumstances this can lead to smaller files than those produced by 'run-length' encoding. The file is also much easier to expand because the compression is based on spatial position.

Another version of the quad-tree is analogous to the one dimensional B-tree, referred to above, which is widely used to index non-spatial data . The index is a tree structure as shown in the figure below. Each 'leaf' will contain anywhere between a certain maximum number of spatial elements and quarter of that number. When a 'leaf' becomes full, it is split again into four quadrants. The quad-tree differs from the B-tree in that with the quad-tree the split is always based on a fixed grid, while with the B-tree the split is based on the actual range of the data. Accessing data, when a co-ordinate range is given, is simply a matter of finding the tree 'leaves' corresponding to that co-ordinate range and getting the elements contained on those 'leaves'.

A third possibility is a two-dimensional representation of the B-tree - similar to the above case except that the 'leaves' will not be based on a regular subdivision of a grid but will be based on the actual distribution of graphic elements. This index would be accessed via a pair of linear indexes on the X and Y co-ordinates.

The quad-tree makes it possible to index lines and polygons. If a line crosses a tree 'leaf', no matter where its end points are, it will be possible to find it quickly from a search point within the leaf.

These concepts are clarified in the figure below.

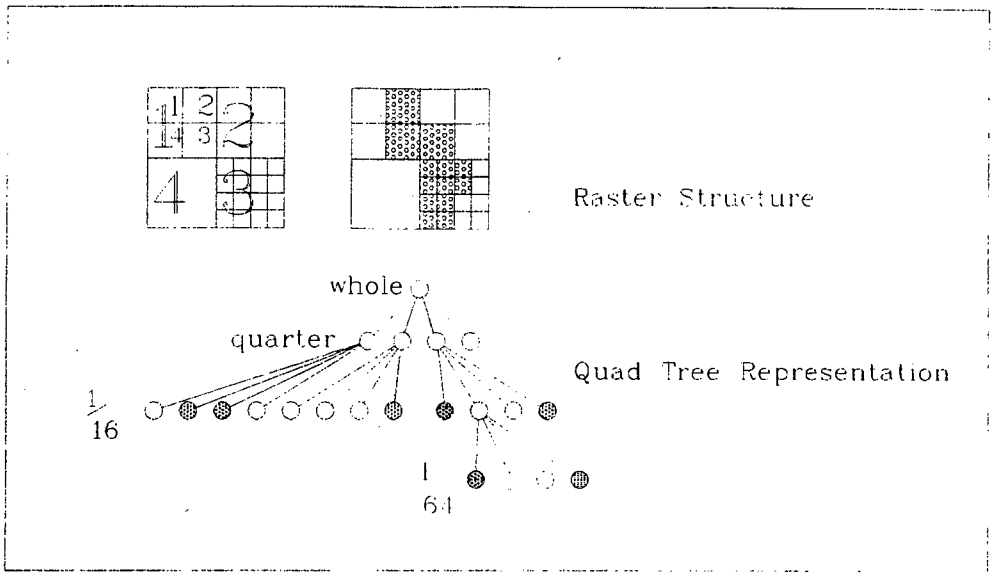


Fig. 5 - 6 Quadtree for Raster Data Compression

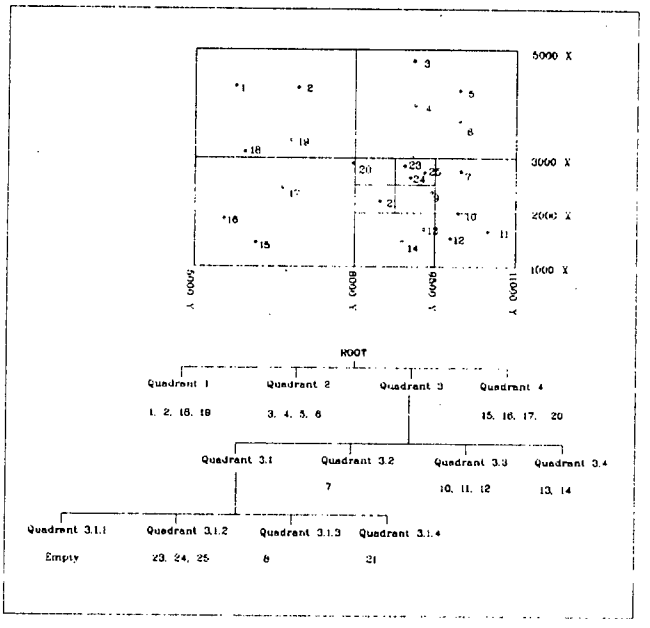


Fig. 5 - 7 Quadtree Index for Points

5.3 Formation of Polygons

It is possible when digitising, or when entering data from the keyboard, to enter polygons directly. Using the digitiser

the perimeter of each polygon can be traced, or from the keyboard the sequence of points forming the polygon can be entered. This method is used by the Ordnance Survey in the U.K., however it requires greater concentration by the operator who must keep in mind which figure he is digitising, where he started, and which figures have been done.

Most organisations simply digitise all the points and lines on the plan and then use the computer to form polygons from the lines and points. Less concentration is required with this method because points do not have to be digitised in any particular order, and if the same point is digitised twice, the computer will only store the first entry. A common method is the so-called 'meat-balls and spaghetti' method whereby lines are digitised at random and form a jumbled mass of lines like spaghetti. Within each polygon a label point is digitised and a lot number or other identifier is typed in - these are the 'meat-balls'. After the computer has converted the 'spaghetti' to polygons, a check is made that each polygon contains one label. Polygons without a label or with more than one label indicate errors in digitising and pin-point the error.

As was mentioned above, data obtained from scanning or remote sensing is in raster form and must be vectorised before it can be integrated into a vector-based LIS. After vectorisation the data is in the form of lines and points (the end points of the lines) and must be polygonised in the same way as digitised data.

An algorithm for converting lines and points into polygons is as follows:-

Identify the points which are nodes (more than two lines meeting at these points)

---

1 Structured Digital Data at OS, Land & Mineral Surveying, March 1988.

Form chains by taking each node in turn, then taking each line from that node in order to form a list of connected lines. Continue until another (or the same) node is reached.

The example which follows refers to Fig. 5-2.

The starting data is the set of points (1 2 3 4 5 6 7 8 9) and the set of lines (1-2,2-3,3-4,4-5,5-1,3-9,9-8,4-8,8-7, 5-7,7-6,6-1).

If a point number appears more than twice in the set of lines it means that that point is a node. From this it can be seen that 1, 3, 4, 5, 7, 8 are nodes.

Node	Lines from Node
1	1-2, 1-5, 1-6
3	3-2, 3-4, 3-9
4	4-3, 4-5, 4-8
5	5-1, 5-4, 5-7
7	7-6, 7-5, 7-8
8	8-7, 8-4, 8-9

Start at 1, line 1-2 connects to 2-3

3 is a node therefore the first chain is 1-2-3, or in terms of the data structure given above:-

Chain 1 **id = 1**
start = 1
end = 3
left = 0

right = 0
points = 1, 2, 3

The lines 1-2 and 2-3 can now be removed from the set of lines being used to form chains. (They are still in the data set but have been transferred from the set of lines to the set of chains).

The next chain starts with the line 1-5. 5 is a node therefore this chain consists of only one line.

This operation is repeated until the set of lines is empty. If the set of lines from nodes is removed from the main (original) set of lines and the latter is still not empty, it means that some point in the remainder must be adopted as a 'pseudonode' and the process repeated. In the example nine chains will be formed which are marked c1 - c9 in the figure.

A polygon may be defined as '**simple**' if it cannot be broken down into any smaller polygons, and 'complex' if it is made up from smaller polygons. e.g. in the figure above polygon 1-2-3-4-5-1 is a simple polygon while 1-2-3-9-8-4-5-1 is a complex polygon. If all the possible closed routes through the chains are followed many complex polygons will be formed whereas the aim of the polygonisation process is to produce only simple polygons.

The next step borrows from 'graph' or 'network' theory. This is an area of computer science which deals with networks of points connected by lines. Algorithms have been developed to determine shortest paths, to determine which points are connected etc. This theory has a wider application in LIS for tasks such as finding the optimum route for transport services or power distribution, but in this case only the 'adjacency list' will be used. The adjacency list is a table in which each node is represented by a row containing all the chains starting from that node.

Node	Chains from Node		
1	(1)	(2) 3	Polygon 1:
3	5	4 1	Chains
4	-4	7 6	c1 c4 c6 -c2
5	-2	-6 8	
7	-8	9 -3	
8	-7	-5 -9	

Fig. 5-8 Adjacency Table

All the routes traversing the network of lines (an un-directed graph in the language of the computer scientist, 'un-directed' because lines may be traversed in either direction) can be determined from this table.

e.g. Starting from row 1 (node 1) one can follow chain 1. Chain 1 appears again in row 2 (node 3) meaning that one can reach node 3 from node 1 by following chain 1. From node 3 one might follow chain 5 which occurs again in row 6 (node 8).

In forming polygons the only paths to be followed are those which form simple polygons. In order to do this the adjacency list has to be formed in a special way, namely as described above except that the chains from each node must be sorted according to the bearing of the first line of each chain. In other words they must be listed in the order in which they

would be encountered going round the node in either a clockwise or an anti-clockwise direction.

In the figure above the chains must be ordered c3, c4, c1, c2 or c4, c3, c2, c1 etc. - the sequence may start anywhere - but **not** c1, c3, c2, c4.

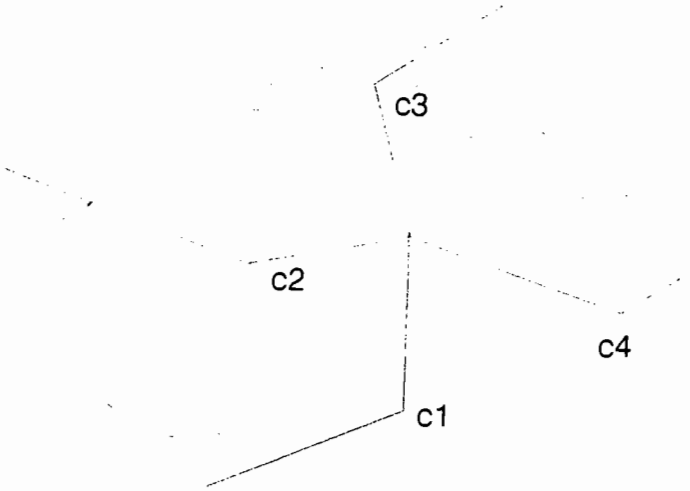


Fig. 5 - 9 Ordering of Chains

Once the adjacency list has been formed in this manner it is a straightforward procedure to find simple polygons in clockwise direction. If one arrives at a node along chain c2 for example, one must leave again on the next chain in an anti-clockwise direction i.e. c1. This chain will be next to the arrival chain in the adjacency list (whether on the right or the left depends on whether the chains are in clockwise or anti-clockwise order).

The chains in the table given in figure 5-7 are already sorted into clockwise order. Form polygons as follows:-

Start on the top row of table 5-7 at the first chain - node 1, chain 1.

Search every other row for chain 1, the sign will be different because one will be viewing the chain from its other end. -1 is found in row 2 which corresponds to node 3.

Chain 1 can now be added to the first polygon.

Next move one column to the left in row 2 of the table (if already in column 1 move to the end of the row). In this case the next chain is 4. Search for -4 which is found in row 3 corresponding to node 4.

Add chain 4 to the first polygon.

Move left in row 3. The next chain is 6. -6 is found in row 4 corresponding to node 5.

Add chain 6 to the first polygon.

Move left in row 4. The next chain is -2. 2 is found in row 1 corresponding to node 1.

Add chain 2 to the first polygon. The polygon is now closed having arrived back at the starting node 1. The perimeter of the polygon is specified by chains c1, c4, c6, -c2.

Having completed one polygon, the polygon number (1) can be placed in the 'right' field of each positive (forward) chain and in the 'left' field of each negative (backward) chain.

To form the second polygon move one column to the right in the top row and repeat the process. The starting point moves progressively along each row and then on to the next row until every element in the table has been tried as a starting point. Obviously every polygon will have as many duplicates in the table as it has nodes. If the 'right' of a chain is not zero it means that the polygon has already been formed and the program can skip to the next point.

5.4 Querying the Spatial Database

Up to this point this thesis has focussed on various aspects of designing and building a database for spatial information. In this final section the problems of accessing and manipulating the data will be considered.

Queries may relate to non-spatial data only, or to a combination of spatial and attribute data. Some of the more common types of query are discussed below.

Attribute Data Only

The query "List the names of owners of properties valued at over R 1 000 000" involves only attribute data and is no deifferent from querying any non-graphic database. It might be expressed in SQL as

"SELECT NAME FROM OWNERSHIP WHERE VALUATION > 1000000"

if there was a table called "OWNERSHIP" with fileds "NAME" and "VALUATION".

A graphic device serves no purpose with this type of query.

Attribute and Spatial Data

Spatial elements can be selected either by identifying them with a graphic input device, or by specifying one or more of their attributes.

These queries fall into two categories - those relating to the attributes of a single spatial element, and those combining or overlaying different spatial elements. In each case the spatial element may be selected by attribute or by graphic input device.

An example of the first type would be "List (or show) the farms growing wheat with an area greater than 1000 ha". This is similar to the attribute only query mentioned above. The mechanics of the query are exactly the same. A table is searched to select the records that meet the given criteria. The difference lies in the answer, a farms is a spatial (polygon) entity and the answer may therefore be given in a graphic form. In this case the answer might be a list of the co-ordinates of corner points of all the farms which meet the

given criteria. However, a screen display or a plot showing all the farms with those satisfying the query shaded in a different colour, would give a much more easily understood answer to the query.

A similar query using graphic input would be "List the owners and crops for the farms selected on the screen". In this case the appropriate data from the polygon attribute table is listed.

The second type of query which involves the overlaying of different graphic elements is more complicated because, in addition to the table searches, geometric computations are required. There are three cases:-

Point in Polygon

The query "List the boreholes and their yield, on Mr van Zyl's farm" is a spatial query. The polygon with the attribute OWNER = 'Mr van Zyl' and DESCRIPTION = 'farm' is found in the polygon table, Next, all the points in the point table with the attribute 'borehole' are tested to see if they are in the selected polygon. To do this the 'point in polygon' test is performed.

The algorithm used to perform the 'point in polygon' test is quite simple. A very long line is taken from the point to be tested to a point well outside the area of interest. The computations are simplified if the line runs parallel to a grid line. This line is then tested against each side of the polygon to see if the two intersect. The number of sides the line intersects with is counted - an odd number means that the point is in the polygon, while an even or zero number means it is outside.

Polygon Overlays

In order to answer a query such as "List the owners of land with soil type I" it is necessary to overlay the soils map on the cadastral map. The overlay creates a new pattern of polygons which inherit the attributes of both their parents. In the figure 5-10 below a soil type map is overlaid on a cadastral map. The resulting map is the result of the 'join' operation between the two databases and consists of polygons which have the attribute of a particular farm number and soil type. From this overlay queries such as "What proportion of farm 1406 consists of Class I soil?" can be answered.

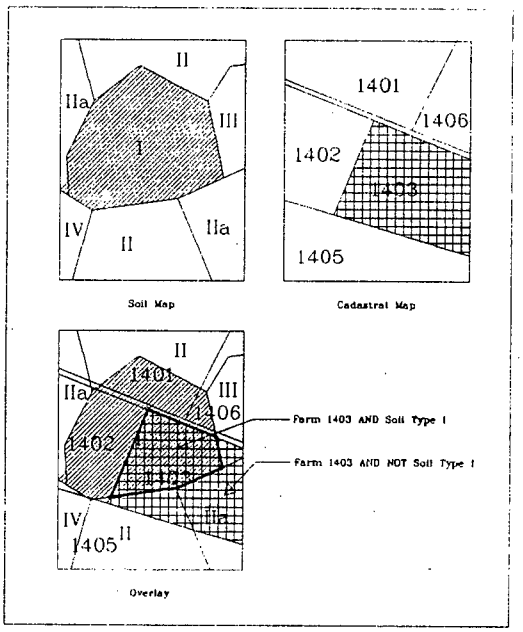


Fig.5-10 Polygon Overlay

In order to compute a polygon overlay the procedure is as follows-

Start with the two line sets which make up the polygons in the coverages to be combined.

Compute all the intersections between lines of the two sets to create a number of new nodes. Existing lines will be split at these intersections to form new lines which will retain the adjoining polygon attributes from their source coverage.

Build a new polygon set from the combined line set. The new polygons will inherit attributes from both parents. These attributes will be taken from the polygon attributes of the left and right polygons for each line in the parent coverages.

Once the overlay coverage has been built, queries can be performed using Boolean operators e.g. "Show the land which is either Farm 1402 or Soil Type II" (XOR), or "Show the land on Farm 1405 which does not have Type I soil" i.e. "Select for Farm No. = 1405 AND NOT (Soil Type = I)"

Problems in creating polygon overlays are the computational effort in testing for and computing a possibly very large number of intersections. An automatic number system also has to be used to identify the new points created by the intersections.

5.5 A Data Model to Describe the Temporal Element in LIS[2]

The need for preserving a historical record of the changes to a cadastral LIS was discussed in 4.11 above. One way to do this to develop a suitable data model.

When developing a data type to represent the 'lot' or land parcel the following points must be taken into account:-

---

2 Price V J S, Modelling the Temporal Element in Land Information Systems, International Journal of Geographical Information Systems, Vol.3 No.3 pp.233-244, July-September 1989.

The lot is dynamic - its area changes with the passage of time as sub-divisions are deducted or re-surveys are carried out.

Every lot will have one or more 'parent' lots from which it inherits certain attributes.

A lot may give birth to one or more 'child' lots as sub-division take place.

The current state of the lot i.e. the boundaries of the remaining extent is needed to generate a graphic display. A 'pick' operation on the display should lead to a lot only when the graphic cursor is placed on the remaining extent of the lot - in other cases it must lead directly to the appropriate child lot.

An essential attribute of each lot is its date of creation so that a map of the cadastre at any point in the past can be built up.

Before proceeding to define a data structure the concept of the 'remainder' or 'remaining extent' of a lot may need some explanation. In some systems whenever a lot is sub-divided, the sub-division will form a new lot, and what is left of the original lot will also be constituted as a new lot i.e. the original lot disappears altogether. This obviously makes sense when a large lot is subdivided into a township or a number of lots more or less equal in size. However if a small piece is cut from a large area, such as when an electricity sub-station site is taken from a large farm, it makes more sense to treat the sub-division as a deduction from the parent lot. This continues to exist, though with a different area and perimeter, and forms the remainder of the parent lot. This appears a more logical way to view a sub-division transaction. It is probably also administratively more straightforward to issue one new title and note a deduction against the original title than to issue two new certificates

of title, especially when the parent is a large complicated figure and the deduction is small. In the South African system the professional land surveyor has the option to leave a remainder or not, according to the circumstances of the case. In the Swiss system the largest portion after subdivision retains the original lot number[3].

Using the syntax of Modula-2 an abstract data type for a lot can be defined as follows:-

```
TYPE
lot = RECORD
     lot_no : LONGCARD (* unique identifier*);
     sides  : LIST (* of 'chains' - original boundary *);
     area   : LONGREAL (* original area *);
     parent : LIST (* of lots *);
     rem    : LIST (* of 'chains' - present boundary *);
     rm_area: LONGREAL (* present area *);
     consol : LONGCARD   (* lot_no of new lot if lot *)
                         (* has been taken into a    *)
                         (* consolidation            *)
     trans  : LIST (* of transactions *);
END;
```

The field 'lot_no' provides the lot with a unique identifier which serves as a primary key and as a link into a standard relational database.

The fields 'sides' and 'rem' define the original boundaries and the present boundaries respectively of the lot. The purpose of including the present boundaries is to enable graphic software to draw the lot in its current state, or to identify it from an input co-ordinate, without the time-consuming process of working through all the sub-divisions. Note that the sides will be defined by 'chains' or lists of lines linking 'nodes' or points where more than two lots meet. Each chain includes as an attribute the lot number of

---

3 Kauter B, The Swiss Cadastre, Proceedings of the Seminar Series on Land Information and the Land Surveyor, Technical Report No.126 p.12, University of New Brunswick Dept. of Surveying Engineering, October 1986.

the lots abutting it to the left and right. In this way the data structure reflects the topological structure.

In a computerised LIS, points must be represented by co-ordinates. Bearings, distances, and areas can be computed from these co-ordinates. However, such computed dimensions may not agree exactly with the dimensions recorded on the survey plan of the lot which has a legal status i.e. legal and mathematical dimensions may not be the same. This is the case in Singapore. There co-ordinates are computed from adjusted survey observations while unadjusted observations are recorded on the plan. Under the computerised system the

plan data is held as attribute data of the lot. The differences between legal and mathematical data are of concern only to land surveyors and appear trivial to others.

The fields 'area' and 'rm_area' are the original and present (legal) areas respectively. These could be readily computed from the sides but in some cases the legal area for transfer may not agree exactly with the computed area. If rm_area = 0 then the lot has been subdivided leaving no remainder. In this case the field 'rem' will be an empty list.

The other case in which a lot ceases to have an active existence is when it is consolidated with one or more lots to create a new one. The field 'consol' shows the number of the new lot. In other cases this field has a value of zero.

The fields 'parent' and 'trans' show the history of the lot. 'parent' is a list of lot numbers showing the origin of the lot. This field will have a nil value if the lot is an original grant from unallocated state land or an existing lot brought into the cadastral system for the first time. If the list contains more than one lot number then the lot is a consolidation, otherwise the single parent property from which it was excised is shown.

The 'trans' field contains a list of transactions which have happened to the lot. This gives a history of the lot. The field comprises a list of transaction records each of which might defined as

```
transaction = RECORD
                date     : ARRAY[0..8] OF CHAR;
                type     : transaction_type;
                trans_id: LONGCARD;
                lot_no   : LONGCARD;
                cancel   : BOOLEAN;
                c_date   : ARRAY[0..8] OF CHAR;
                figure   : LIST (* of points *);
            END;

transaction_type = (transfer,CCT,CRT,mortgage,caveat,.
```

Each element in the list records a transaction. The field 'trans_id' provides a key to a transaction file where full details of each transaction are recorded. If the transaction is the transfer of a portion of the lot, the 'lot_no ' field will show the number of the sub-division which has been deducted.

Certain transactions such as the registration of caveats, mortgages, servitudes, leases, charges, etc. either lapse or are subsequently canceled; the fields 'cancel' and 'c_date' record the fact and date of cancellation. The field 'figure' shows the area of the land affected by transactions which do not result in the creation of a new lot but which affect only part of the land.

Having created the model it is necessary to consider the permissible operations on it. These fall into two categories - updates and queries.

All updates result from the registration of a deed of one form or another. They can be divided into three classes, those that create a new lot, those that create a new right over an existing lot, and those that cancel such rights. The following procedures might be defined to handle these cases. It is assumed that these procedures are carried out concurrently with the addition of the transaction or Deed to the transaction database.

```
PROCEDURE Create(lot_no: LONGCARD (* new lot No.       *);
                 trans : transaction_type (* Grant, CRT,
                                          transfer  *);
                 t_date: ARRAY[0..8] OF CHAR; (* date  *);
                 t_no  : LONGCARD;  (*transaction no.  *);
                 parent: LIST (* of lot numbers        *);
                 figure: LIST (* of points             *));
```

This procedure creates a new lot record. The record of the parent lot is amended to reflect its new boundaries and area and the transaction details are added to the appropriate field. Registered interests such as servitudes and leases which affect the area subdivided must be carried forward to the new lot record.

```
PROCEDURE Update(lot_no: LONGCARD;
                 trans  :transaction_type (* Caveat,Lease*);
                                          (* Servitude   *);
                 t_date : ARRAY[0..8] OF CHAR;(* date    *);
                 t_no   : LONGCARD;
                 figure : LIST (* of points             *));
```

This procedure handles transactions which do not result in the creation of a new lot.

```
PROCEDURE Cancel(lot_no: LONGCARD;
                 t_date: ARRAY[0..8] OF CHAR;(* date    *);
                 t_no  : LONGCARD;
                 figure: LIST (* of points             *));
                 old_no: LONGCARD;
```

Transactions such as leases, mortgages and caveats normally come to an end either through the effluxion of time or

through the action of the parties to the agreement. This procedure marks the original entry as cancelled as well as recording the cancellation as a new transaction.

Procedures can also be written to answer common queries such as "Who was the registered owner on such and such a day?". In this case the procedure would search through the transaction list (ordered by date) of the relevant lot for the latest transfer of the whole property, before the specified date. The entry in the transaction file corresponding to this transfer would supply the owner's name, as well as other data such as the purchase price.

This approach to data modeling for cadastral land information systems outlined above, represents a logical and structured approach and has two advantages compared with current practice. Firstly, the problem of archiving and recalling historic data is removed - the history of a lot is implicit in the data structure, and second, the job of the applications programmer is simplified due to the use of abstract data types. Better quality software should therefore result. The method is also flexible and can easily be adapted to different cadastral systems.

5.5.1 An Example

In the figure below the position is as follows - lot 1 has been created by a grant of state land. Lots 2 and 3 are subdivisions of lot 1. Lot 3 is itself subdivided into lot 4 and a remainder. Lots 4 and 2 are consolidated to form lot 5.

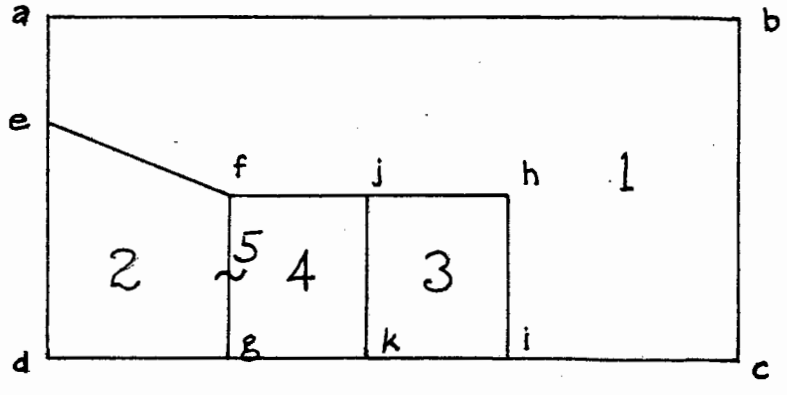


Fig. 5 - 11

The steps are:

```
Create(1,grant,'13/2/88',1,'','a b c d');
```

lot 1

```
lot_no : 1
sides  : a b c d
area   : 10.0000
parent : NIL
rem    : a b c d
rm_area: 10.0000
consol : 0
trans  : 1
```

transaction 1

```
date    :13/2/88
type    :grant
trans_id:1
lot_no  :
cancel  :
c_date  :
figure  :
```

```
Create(1,transfer,'17/3/88',2,'1','e f g h');
```

lot 1

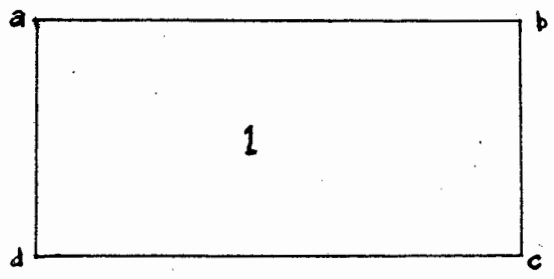
```
lot_no : 1
sides  : a b c d
area   : 10.0000
parent : NIL
rem    : a b c g f e
rm_area: 9.0000
consol : 0
trans  : 1 2
```

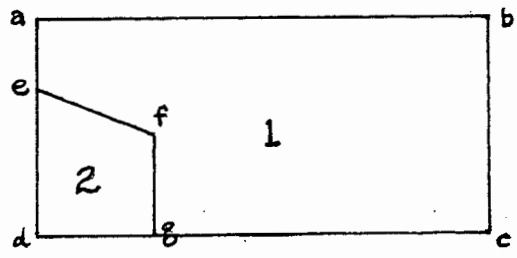
lot 2

```
lot_no : 2
sides  : e f g d
area   : 1.0000
parent : 1
rem    : e f g d
rm_area: 1.0000
consol : 0
trans  : 2
```

transaction 2

```
date    :17/3/88
type    :transfer
trans_id:2
lot_no  :2
cancel  :
c_date  :
figure  :
```





```
Create(3,transfer,'21/4/88',3,'1','f h i g');
```

<u>lot 1</u>

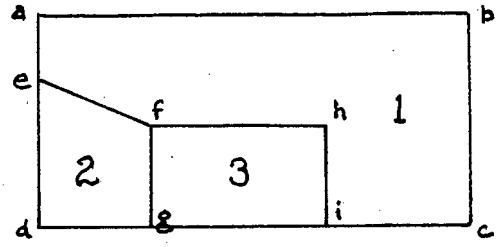
```
lot_no : 1
sides  : a b c d
area   : 10.0000
parent : NIL
rem    : a b c i h f e
rm_area: 8.0000
consol : 0
trans  : 1 2 3
```

<u>lot 3</u>

```
lot_no : 3
sides  : f h i g
area   : 1.0000
parent : 1
rem    : f h i g
rm_area: 1.0000
consol : 0
trans  : 3
```

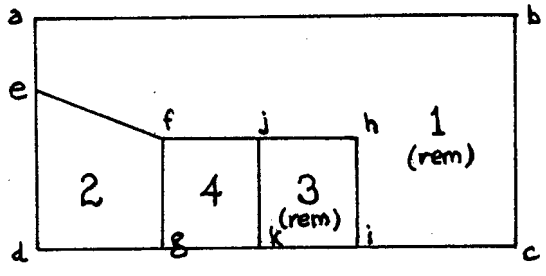
<u>transaction 3</u>

```
date    :21/4/88
type    :transfer
trans_id:3
lot_no  :3
cancel  :
c_date  :
figure  :
```




```
Create(4,transfer,'25/5/88',4,'3','f j k g');
```

lot 1 is not affected



lot 3

```
lot_no : 3
sides  : f h i g
area   : 1.0000
parent : NIL
rem    : j h i k
rm_area: 0.5000
consol : 0
trans  : 3 4
```

lot 4

```
lot_no : 4
sides  : f j k g
area   : 0.5000
parent : 3
rem    : f j k g
rm_area: 0.5000
consol : 0
trans  : 4
```

transaction 4

```
date    :25/5/88
type    :transfer
trans_id:4
lot_no  :4
cancel  :
c_date  :
figure  :

Create(5,CCT,'29/6/88',5,'4 2','e f j k g d');
```

lots 1 and 3 are not affected

lot 2

```
lot_no : 2
sides  : e f g d
area   : 1.0000
parent : NIL
rem    : e f g d
rm_area: 1.0000
consol : 5
trans  : 2 5
```

lot 4

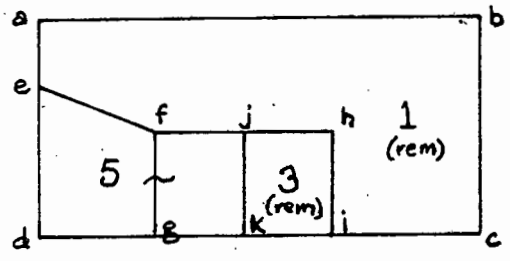
```
lot_no : 4
sides  : f j k g
area   : 0.5000
parent : 3
rem    : f j k g
rm_area: 0.5000
consol : 5
trans  : 4 5
```

transaction 5

date :29/6/88
type :CCT
trans_id:5
lot_no :5
cancel :
c_date :
figure :

lot 5

lot_no : 5
sides : e f j k g d
area : 1.5000
parent : 2 4
rem : e f j k g d
rm_area: 1.5000
consol : 0



REFERENCES

A Course of Lectures Prepared for Third Year Survey Technicians (Cadastral) Studying for the National Diploma Examination, Vol.1 History and Legislation Relating to Land Surveying in South and South West Africa, Director General Surveys, Mowbray, 1961.

A National Standard for the Exchange of Digital Geo-referenced Information, National Research Institute for Mathematical Sciences, Pretoria, 1987.

Ben-Zvi J, The Time Relational Model, PhD thesis, Dept. of Computer Science, University of California,Los Angeles, 1982.

Burrough P A, Principles of Geographical Information Systems for Land Resource Assessment, Oxford University Press.

Cairns M, The Archival Role in Establishing the Ownership of Land, History of Surveying and Land Tenure in South Africa, Vol.1 pp.89-98, University of Cape Town, 1984.

Cartographic Data Standards (NCDCDS), 1986.

Codd E F, A Relational Model of Data for Large Shared Data Banks, Comm. ACM 13 (6/6/70) p.377-387.

Codd E F, A Data Base Sublanguage founded on the Relational Calculus' Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control.

Couesnou T, Laurent D, Motet S, The Geo-Graph Simulation System: Towards Dynamic Use of a Geomatic Data Base, Advanced Computer Graphics: Proceedings of Computer Graphics in Tokyo '86, Springer-Verlag, Tokyo 1986.

Dale P F, Cadastral Surveys within the Commonwealth, HMSO, 1976.

Dale P, De Simone M, Vector-Raster-Vector Conversions, Land & Minerals Surveying, March 1986.

Dale P, De Simone M, The Automatic Recognition of Features in Digital Maps, Land & Minerals Surveying, June 1986.

Date C J, A Guide To The SQL Standard,Addison-Wesley, 1987.

Dirdal P, Data Acquisition: the Case for Raster, Land & Mineral Surveying, January 1988.

Dowson & Sheppard, Land Registration, HMSO 1952.

Epstein E, Duchesneau T, The Use and Value of a Geodetic Reference System, Report for US Federal Geodetic Control Committee, 1984

Federal Geographic Exchange Format, Federal Interagency Co-ordinating Committee on Digital Cartography, 1986.

Fletcher L N, The Integrated Survey Systems and Modern Mapping in New South Wales, Proceedings of URPIS-ONE, 1973.

Frank A U,LOBSTER: Combining Database Management and Artificial Intelligence Techniques to Manage Land Information, Proceedings of FIG XVIII International Congress,Vol.3 pp.2-13, Toronto 1988.

Frank A U, Requirements for Database Systems Suitable to Manage Large Spatial Databases, Proceedings of the International Symposium on Spatial Data Handling, Zurich, 1984.

Gad Ariav, A Temporally Oriented Data Model, ACM Transactions on Database Systems, Vol.11 No.4 pp.499-527, December 1986.

Ginsberg S and Tanaka K, Computation-Tuple Sequences and Object Histories, ACM Transactions on Database Systems, Vol.11 No.2 pp.186-212, June 1986.

Hamilton A C and Williamson I P, A critique of the F.I.G. definition of "Land Information System", F.I.G. International Symposium on LIS, Edmonton, 1984.

Hart A L,The Politics of Land Information Systems, FIG XVII International Congress, Toronto, 1986, Commission 3 - Land Information Systems, pp.267-278.

Hawkey W N, Land Information New Zealand: The Keys to Progress, Regional Seminar on LIS in Pacific Rim, July 1987.

Hodges D J and Cooper S M, Computer-Aided Digitisation of Maps and Plans, Land & Mineral Surveying, June 1988.

Hughes J G, Database Technology A Software Engineering Approach, Prentice-Hall 1988.

Humphreys B, The Crisis in Land Management, National Seminar on Land Information Systems, Kuala Lumpur, 1984

Hunter G J, Non-Current Data and Geographical Information Systems. A Case for Data Retention, International Journal of Geographical Information Systems, Vol.2 No.3 pp.281-286, July-September 1988.

Interchange of Feature Coded Digital Mapping Data: Australian Standard 2482-1981, Standards Association of Australia, 1981.

Intergraph Corporation, Standard Interchange Format (SIF) Command Language Implementation Guide, Intergraph Corporation Document No. DIXD7330, Huntsville, Alabama, 1986.

Kauter B, The Swiss Cadastre, Proceedings of the Seminar Series on Land Information and the Land Surveyor, Technical Report No.126 p.12, University of New Brunswick Dept. of Surveying Engineering, October 1986.

Langefors B, Infological Models and Information Users' Views, Information Systems No.5 pp.17-32, 1980.

Lyons K J, Eden E J and Chen C C, Some Thoughts on Designing Land Information Systems to Fit into a Country's Social and Administrative Framework, Regional Seminar on LIS in Pacific Rim, July 1987.

Marrao E, TIGRIS: A Third Generation GIS, Proceedings of the Ninth Conference of South African Surveyors, Paper No. 9.2, March 1989.

Martin J. Principles of Data-Base Management, Prentice-Hall 1976.

McKenzie E, Bibliography: Temporal Databases, ACM SIGMOD Rec.15 No.4 pp.40-52, December 1986.

Nyerges T, Testing the Interim Proposed Standard for Cartographic Data Exchange, Digital Cartographic Data Standards: A Report on Evaluation and Empirical Testing, Report No.7, National Committee for Digital

Osborn S L and Heaven T E,The Design of a Relational Database System with Abstract Data Types for Domains, ACM Transactions on Database Systems, Vol.11 No.3 pp.357-373, September 1986.

Price V J S, Modelling the Temporal Element in Land Information Systems, International Journal of Geographical Information Systems, Vol.3 No.3 pp.233-244, July-September 1989.

Rapaport M, Object-oriented data bases: The next step in DBMS evolution, Computer Language, Vol.5 No.10 pp.91-98, October 1988.

Research Advisory Information Group, Land & Minerals Surveyor (L & MS) February 1988.

Rhind D W,Geographical & Land Information Systems: Their Relationship, Shortcomings and Future Prospects, Conference of Southern African Surveyors, March 1989.

Rhind D W and Green N P A, Design of a Geographical Information System for a heterogeneous scientific community, Geographical Information Systems, Vol.2 No.2 June 1988.

Rokos D K, The Contribution of an Integrated Cadastral LIS for Decision Making, FIG XVII International Congress, Toronto, 1986, Commission 3 - Land Information Systems, pp.150-164.

Sharman D, H M Land Registry & Ordnance Survey Digital Mapping, Land & Minerals Surveying, January 1986.

Snodgrass R, The Temporal Query Language TQuel, ACM Transactions on Database Systems, Vol.12 No.2 pp.247-298, June 1987.

Structured Digital Data at OS, Land & Mineral Surveying, March 1988.

The National Transfer Format Release 1.0, Ordnance Survey, Southampton, 1987.

Thomas D,What's in an Object?, Byte Magazine, p.231 March 1989.

Tsichritzis D C and Lochovsky F H, Data Models, Prentice-Hall, 1982.

U.S. National Bureau of Standards, Initial Graphic Exchange Specification (IGES) Version 2.0, U.S. Department of Commerce, National Technical Information Centre, Publication No.PB83-127448, Washington D.C., 1983.

Uhlenbruck H M, The Application of Expert Systems in Geographical Information Systems, University of New Brunswick Department of Surveying Engineering Technical Report No.138, June 1988.

Wiener and Sincovec, Software Engineering with Modula-2 and Ada, Wiley, 1984.

Woodsford P, Data Acquisition: the Case for Vector, Land & Mineral Surveying, January 1988.

Hj Zainal Abidin, Progress and Problems of Computerisation of Land Administration in Peninsular Malaysia, Regional Seminar on LIS in Pacific Rim, July 1987.

Appendix A

Software

The demonstration software developed to illustrate the concepts discussed in this thesis has been developed in the Modula-2 programming language. The compiler used was the JPI 'Topspeed' Modula-2.

The software comprises:-

a). A main module called THESIS which contains co-ordinate geometry procedures for the createtion of a vector-based spatial database, graphics routines, routines to link lines into chains and to form polygons. This module depends on

1). A set of database routines which provides variable length indexed files. These routines are marketed by Messrs PMI of Portland, Oregon under the name 'Repertoire'.

2). Four modules **LINES**, **COORD**, **POLYGONS**, and **COVERS**. These modules define the main data structures and handle all the basic data management operations on these structures such as storing, retrieving, updating and searching for data elements.

3). A number of auxiliary modules for input/output, handling of strings, lists, queues etc.

b). A main module **MODISTRU** which modifies the database structure as required to accomodate fields for attribute data.

c). A main module **DE** which handles the entry of data into the attribute fields of the database.

The definition part of the most important modules are given below with notes on the function performed by each procedure.

• DEFINITION MODULE coord;

This module provides procedures for accessing and manipulating the co-ordinates of points.

```
FROM FIO IMPORT File;
FROM NdxTypes IMPORT NdxFileType;
TYPE
    leftright = (left,right,unknown);
    point = RECORD
        id: CARDINAL;
        north,east: LONGREAL;
    END;
    coordset = NdxFileType;
```

A set of points is defined as a '**coordset**' which is an indexed file imported from the 'Repertoire' system (see above). A very strong point in favour of languages such as Modula-2 is that the actual structure of the data type '**coordset**' is irrelevant outside this module. A group of points could be represented by an array in memory or a dBase file or a linked list and it would only be necessary to rewrite the '**coord**' implementation module.

```
    id = CARDINAL;
```

Each point has a unique identifier used as a label. In a serious application an alpha-numeric label would be used, but for a demonstration the use of a cardinal number simplifies programming. This applies also to chain and polygon identifiers.

```
VAR
    CHECK: BOOLEAN;
```

When this variable is **TRUE** a check is made before over-writing existing co-ordinate values.

PROCEDURE WrCoord(f: File; p: point);

This writes a co-ordinate to a disc file, to the screen if 'f' is equal to the standard DOS device 'scr' or to a printer if 'f' is equal to 'prn'.

PROCEDURE WrCoordList(f: File; c: coordset);

This uses the procedure above to write all the co-ordinates to the selected device.

PROCEDURE SaveCoords(fn: ARRAY OF CHAR; c: coordset);

This procedure uses 'WrCoordList' to write the set of co-ordinates to a disc file with the name contained in the variable 'fn'.The file is in text form but retains six figures after the decimal point.

PROCEDURE LoadCoords(fn: ARRAY OF CHAR; c: coordset);

This will read co-ordinates from a text file on disc and add them to the co-ordinate database.

PROCEDURE FindCoord(VAR n: CARDINAL; VAR c: coordset):point;

A function which returns a point with the point ID of 'n'. If the point is not found a zero ID is returned.

PROCEDURE AddCoord(p: point; c: coordset);

Point 'p' is added to the co-ordinate set. If a point with the same point ID as 'p' is already in the set and the variable CHECK is TRUE a check is made with the operator before over-writing the old value.

PROCEDURE RemoveCoord(p: point; c: coordset);

A point is deleted.

PROCEDURE CreateCoordSet(VAR x: coordset);

A new co-ordinate set i.e. database file is created.

PROCEDURE EraseCoordSet(VAR x: coordset);

The database file is deleted.

PROCEDURE EmptyCoordSet(VAR x: coordset);

The database file is not deleted but all the points in it are removed.

PROCEDURE GetFirstCoord(VAR c: coordset):point;

This is a function which returns the first co-ordinate in the set. For certain operations such as listing or searching for the nearest point to another point it is necessary to process the file sequential.

PROCEDURE GetNextCoord(VAR c: coordset; p: point): point;

This returns the point after point 'p' in a sequential traversal through the file.

PROCEDURE MaxXY(VAR mx,my: point; VAR c: coordset);

This procedure finds the maximum X and Y values in the file. This might be used to create a window for plotting.

PROCEDURE MinXY(VAR mx,my: point; VAR c: coordset);

Finds the minimum values in the file.

PROCEDURE FindNearestToPoint(p: point; VAR c: coordset): point;

This function returns the point in the file nearest to the given point 'p'.

PROCEDURE FindNearestToLine(a,b: point; VAR x: leftright; VAR c: coordset): point;

This function returns the point nearest to the line joining point 'a' to point 'b'. If the function is called with 'x' set to '**unknown**' the absolute nearest point to the line will be returned and 'x' will be set to '**left**' or '**right**' depending on which side of the line, looking from 'a' to 'b', the point lies.

If the function is called with 'x' set to 'left' or 'right' then the point returned will be the nearest on that particular side of the line.

This function is useful in building triangles for the creation of digital terrain models.

PROCEDURE rectpol(dx,dy: LONGREAL; VAR r,s: LONGREAL);

Rectangular co-ordinates are converted to polar co-ordinates.

PROCEDURE jn(pt1,pt2: CARDINAL;VAR data: coordset; VAR r,s: LONGREAL);

The '**rectpol**' function is used to compute the bearing and distance between 'pt1' and 'pt2'.

PROCEDURE sameside(p1,p2,p3,p4: point; VAR c: coordset): LONGREAL;

This function returns a positive value if points 'p3' and 'p4' line on the same side of line 'p1'-'p2', a negative value if they are on opposite sides of the line, and 0 if one of the points is on the line.

PROCEDURE intersect(p1,p2,p3,p4: point; VAR c:coordset): BOOLEAN;

The previous routine is used to determine whether line 'p1'-'p2' intersects line 'p3'-'p4'. This routine is used to test if a point is inside a polygon. A line joining the point to a very distant point is checked for intersection with each side of the polygon. An odd number of cuts indicates that the line is in the polygon.

PROCEDURE CloseCoordSet(VAR cs: coordset);

This procedure closes the database file before quitting the program.

END coord.

• DEFINITION MODULE lines;

The line module provides procedures for storing and retrieving lines. Most have an analogy with the '**coord**' functions and the names are chosen to make this clear.

```
FROM FIO IMPORT File;
FROM GenLists IMPORT GenList;
TYPE
    line =RECORD
    pt1, pt2: CARDINAL;
    END;
```

The line comprises two end points.

lineset = GenList;

Lines are contained implicitly in '**chains**' so it was decided not to use an indexed file in which to store them. Instead a linked list is used. Note that one of the big advantages of a modular programming system is that '**lineset**' could be changed to a dBaseIII, Repertoire, or any other type of data structure simply by changing the implementation modules '**lines**'. No other part of the program would be affected.

PROCEDURE WrLine(f: File; p: line);

This procedure writes the end points of line 'p' to device 'f'.

PROCEDURE WrLineList(f: File; c: lineset);

Lists complete lineset 'c' to device 'f'.

PROCEDURE SaveLines(fn: ARRAY OF CHAR; c: lineset);

Lines are listed in text form to the file with the name contained in the variable 'fn'.

PROCEDURE LoadLines(fn: ARRAY OF CHAR;VAR c: lineset);

Lines will be loaded from a disc file and added to the lineset 'c'.

PROCEDURE FindLine(p1,p2: CARDINAL; VAR c: lineset):line;

If the line 'p1p2' or 'p2p1' is in the lineset 'c' this line is returned by the function. If it is not found a line with both end points equal to zero is returned.

PROCEDURE AddLine(p: line;VAR c: lineset);

Line 'p' is added to lineset 'c'.

PROCEDURE RemoveLine(p: line;VAR c: lineset);

Line 'p' is removed from lineset 'c'.

PROCEDURE CreateLineSet(VAR x: lineset);

A new (empty) list is created.

PROCEDURE EraseLineSet(VAR x: lineset);

Lineset 'x' is deleted and the memory it was using is released.

PROCEDURE EmptyLineSet(VAR x: lineset);

Lineset 'x' is emptied but the list is not disposed of i.e. the memory allocated to it is not freed.

PROCEDURE GetFirstLine(VAR c: lineset):line;

Returns the first line in the line set.

PROCEDURE GetNextLine(VAR c: lineset; p: line): line;

Returns the line immediately following line 'p'.

PROCEDURE LineEqual(a,b: line):BOOLEAN;

Lines are regarded as undirected, therefore line AB will equal line BA.

PROCEDURE LinesFrom(l: CARDINAL; VAR ls: lineset): lineset;

This function returns a lineset containing all the lines meeting at the point with ID equal to 'l'. It is used in building chains and polygons.

PROCEDURE CopyLineSet(VAR a,b: lineset);

The 'lineset' variable is actually a 'pointer' i.e. the address of the location in the computer memory where the lines are stored. It is permissible to use the assignment function a : = b to assign the value of 'a' to 'b' but the result will be that 'b' points to the same set of data as 'a'. This procedure actually creates another copy of the data at a different memory location. This data can then be manipulated without disturbing the original data.

END lines.

• DEFINITION MODULE polygons;

This module deals with the formation, storage, and retrieval of chains and polygons.

```
IMPORT FIO;
FROM lines IMPORT line,lineset;
FROM coord IMPORT coordset,point;
FROM GenLists IMPORT GenList;
FROM NdxTypes IMPORT NdxFileType;
TYPE
    idlist = GenList;
    chain = RECORD
    id : CARDINAL;
    start, end : CARDINAL;
    left,right : CARDINAL;
    points : idlist;
    END;
```

The '**start**' and '**end**' fields of the chain allow one to see where a chains starts and finishes without resorting to manipulation of the list '**points**'.

```
    chainlist = NdxFileType;
```

polygonset = NdxFileType;

These two data types are actually indexed files of the same type as '**coordset**'.

polygon = RECORD

id: CARDINAL;

sides: idlist;

inside: idlist; (* list of polygons *)

label: ARRAY[0..8] OF CHAR;

labx,laby: LONGREAL;

END;

Each polygon has a unique identifier 'id'. The field '**sides**' is list of chains, and '**inside**' is a list of polygons inside this one. Initially the polygon ID is placed in the label field, and the co-ordinates of the label position '**labx**' and '**laby**' are given the value of the centroid of the polygon.

PROCEDURE WrId(F:FIO.File; p: INTEGER);

Writes an integer value to device 'f'.

PROCEDURE WrIdList(f: FIO.File; VAR c: idlist);

Writes an '**idlist**' which is a list of integers.

PROCEDURE FindId(p1: CARDINAL; VAR c: idlist):BOOLEAN;

This function returns TRUE is 'p1' is a member of the list 'c'.

PROCEDURE GetFirstId(VAR c: idlist):CARDINAL;

Returns the first ID in list 'c'.

PROCEDURE GetNextId(VAR c: idlist; p: CARDINAL): CARDINAL;

Returns the ID in list 'c' following 'p'.

PROCEDURE RemoveId(p: CARDINAL; VAR ptlist: idlist);

Removes 'p' from idlist 'ptlist'.

PROCEDURE AddId(p: CARDINAL; VAR ptlist: idlist);

This procedure add ID 'p' to list 'ptlist'. At an early stage in the design a decision was made not to allow duplicate entries in an 'idlist'. This subsequently caused a number of problems.

PROCEDURE CreateIdList(VAR x: idlist);

Create a new idlist.

PROCEDURE EraseIdList(VAR x: idlist);

Erase an idlist. Using dynamic memory management memory can be allocated for an object when needed and subsequently reclaimed for another use.

PROCEDURE SaveIdList(n: ARRAY OF CHAR; VAR c: idlist);

Write an idlist to a text file. The idist is simply a list of point numbers or ID's.

PROCEDURE LoadIdList(n: ARRAY OF CHAR; VAR c: idlist);

Read a sequence of point ID's from a disc file into an idlist.

PROCEDURE IsNode(p: CARDINAL; VAR x: lineset): BOOLEAN;

Returns **TRUE** if the point with the ID 'p' is a 'node' i.e. if more than two lines meet at point 'p'.

PROCEDURE IdListEqual(VAR p,q: idlist): BOOLEAN;

Returns **TRUE** if the two idlists 'p' and 'q' are equal. They are regarded as being equal if they contain the same points in the same order or in the reverse order. i.e. list 1 2 3 4 is regarded as equal to list 4 3 2 1.

PROCEDURE InitChain(VAR x: chain);

This procedure initialises a 'chain' by setting numeric fields to zero and creating an empty idlist for the points.

PROCEDURE WrChain(F:FIO.File; p: chain);

Write chain 'p' to device 'F'

PROCEDURE WrChainList(F:FIO.File; p: chainlist);

PROCEDURE ChainEqual(p,q: chain): BOOLEAN;

This function returns **TRUE** if chains 'p' and 'q' are equal. They are considered equal if their point lists are equal.

PROCEDURE FindChain(VAR p1: chain; VAR c: chainlist):BOOLEAN;

This function returns **TRUE** if chain 'p1' is found in chainlist 'c'.

PROCEDURE GetFirstChain(VAR c: chainlist):chain;

This function returns the first chain in a '**chainlist**'. If the chainlist is empty a chain with ID = 0 is returned.

PROCEDURE GetNextChain(VAR c: chainlist;VAR p: chain): chain;

This function returns the chain following chain 'p' in chainlist 'c'. If 'p' is the last chain then a chain with ID equal to zero is returned.

PROCEDURE RemoveChain(VAR p: chain;VAR ptlist: chainlist);

This procedure removes chain 'p' from chainlist 'ptlist'. If the chain is not found in the list then no action is taken.

PROCEDURE AddChain(VAR p: chain;VAR ptlist: chainlist);

Chain 'p' is added to chainlist 'ptlist'. If the chain is already in the list it is no added i.e. no duplicate chains are allowed.

PROCEDURE CreateChainList(VAR x: chainlist);

PROCEDURE EraseChainList(VAR x: chainlist);

PROCEDURE EmptyChainList(xl: chainlist);

PROCEDURE SaveChainList(n: ARRAY OF CHAR;VAR c: chainlist);

Chainlist 'c' is written to an ASCII file with name 'n'.

PROCEDURE LoadChainList(n: ARRAY OF CHAR;VAR c: chainlist);

The ASCII file 'n' is loaded into chainlist 'c'.

PROCEDURE MakeNodeList(VAR l: lineset): idlist;

This function returns an idlist containing the point ID's of all points which are nodes. i.e. points at which three or more lines meet.

PROCEDURE GetNextSide(l: line; VAR ls: lineset): line;

THis function returns the line which links to the end of line 'l'. l.pt2 will be the same as pt1 for the line returned by the function. This function is used to link together lines between nodes to form chains.

PROCEDURE MakeChain(id: CARDINAL; s: line; VAR ls: lineset; VAR nl: idlist;VAR x: chain);

This procedure builds a chain with line 's' as its first line. The chain 'x' is built on to the line 'l' using the lines in lineset 'ls'. It terminates when the end of a line added is found in the node list 'nl'. The chain must be specified by its first line rather than by its two end points because several diferent chains might link the same two nodes.

PROCEDURE MakeChainList(VAR ls: lineset;VAR xl:chainlist);

Chainlist 'xl' is formed by using the **MakeChain** procedure repeatedly until all the lines in lineset 'ls' have been used. Some lines may not be linked to nodes e.g. a single closed polygon has no nodes as defined above. In this case one point from the lineset will be designated as a 'pseudo-node'. If there are still lines left in lineset 'ls' after all the chains from this point have been built, another 'pseudo-node' is created. This continues until linset 'ls' is empty.

PROCEDURE GetChain(p: CARDINAL; VAR cs: chainlist):chain;

This function returns the chain with chain ID 'p'. If the chain is not found an empty chain with ID = 0 is returned.

PROCEDURE ChainsFrom(l: CARDINAL; VAR cs: chainlist;VAR data: coordset): idlist;

This function has a very important role in polygonisation. It returns an idlist list containing the ID's of all the chains in the chainlist 'cs' originating at the point with ID 'l'. In order to extract only simple polygons these chains must be ordered according to the bearing of their first lines.

PROCEDURE ReverseChain(x: chain; VAR y: chain);

This procedure places the reverse of chain 'x' in varaible 'y'. Reversing a chain means interchanging the 'start' and 'end' fields, swapping the 'left' and 'right' fields, changing the sign of the chain ID, and reversing the 'points' list.

PROCEDURE MakePolygons(VAR polys: polygonset; VAR cs: chainlist;VAR data: coordset);

The actual formation of polygons depends on an internal data structure called an 'adjacency table' which is explained in the text of the thesis. An internal procedure '**MakeAdjacent**' takes a '**chainlist**' and a '**coordset**' and forms an adjacency table. The structure of the table depends on the position of the nodes as well as the chains, thus the '**coordset**' containing the co-ordinates on the nodes is an essential input to the procedure.

Having formed the adjacency table it is a straightforward operation to extract the polygons.

PROCEDURE ListPolygon(F: FIO.File;VAR p: polygon; VAR cs: chainlist; VAR data: coordset):LONGREAL;

PROCEDURE ListPolygons(F: FIO.File; VAR p: polygonset; VAR cs: chainlist; VAR data: coordset);

PROCEDURE InsidePolygon(p: point; VAR py: polygon; VAR ch: chainlist; VAR cs: coordset): BOOLEAN;

This function returns **TRUE** if point 'p' is inside polygon 'py'. Variables 'ch' and 'cs' are required to define the polygon. In revising the software these variables would be replace by a single '**cover**' variable. If 'p' is not in 'py' **FALSE** is returned.

PROCEDURE WhichPolygon(pt: point;VAR ps: polygonset; VAR ch: chainlist; VAR cs: coordset):CARDINAL ;

This function returns the polygon-ID of the polygon in the '**cover**' composed of 'ps', 'ch', and 'cs' in which point 'pt' is located.

PROCEDURE FindPolygon(n: CARDINAL; VAR poly: polygon; VAR c: polygonset);

The polygon with polygon-ID 'n' is returned in the variable 'poly'.

PROCEDURE AddLabel(VAR p: polygon; lab: ARRAY OF CHAR; x,y: LONGREAL; VAR plst: polygonset);

This procedure replaces the fields 'label', 'labx', 'laby' in the polygon 'p' in **polygonset** 'plst' with 'lab','x' and 'y'.

PROCEDURE SavePolygons(n: ARRAY OF CHAR;VAR c: polygonset);

Polygons are written in ASCII format to file 'n'.

PROCEDURE LoadPolygons(n: ARRAY OF CHAR;VAR c: polygonset);

Polygons are loaded to **polygonset** 'c' from ASCII file 'n'.

PROCEDURE CreatePolygonSet(VAR ps: polygonset);

PROCEDURE ErasePolygonSet(VAR ps: polygonset);

PROCEDURE EmptyPolygonSet(xl: polygonset);

PROCEDURE PolygonToString(VAR p: polygon; VAR s: ARRAY OF CHAR; VAR cs: chainlist);

This procedure places all the points or vertices comprising polygon 'p' into string 's'. The actual polygon contains only chains so the points have to be extracted from the chains.

PROCEDURE ClosePolygonSet(VAR ps: polygonset);

Close the file.

PROCEDURE CloseChainList(VAR cl: chainlist);

PROCEDURE WrPolygon(F:FIO.File; p: polygon);

PROCEDURE WrPolygons(F:FIO.File; p: polygonset);

PROCEDURE ChainsToLineSet(VAR cl: chainlist; VAR ls: lineset);

This is the inverse operation the '**MakeChainList**' above. The chains are split up into their individual components.

END polygons.

DEFINITION MODULE covers;

The '**cover**' is a data structure comprising points, lines, chains and polygons. It groups together the elements which are inter-dependant and logical belong together. One of the weaknesses of this program is that the concept of a 'cover' as a data element was introduced into the design at a relatively late stage.

```
FROM coord IMPORT coordset,point;
FROM lines IMPORT lineset;
FROM polygons IMPORT chainlist,polygonset,polygon;
TYPE
    cover = RECORD
    cs: coordset;
    ls: lineset;
    ch: chainlist;
    ps: polygonset;
    END;
VAR
    CoverName:ARRAY[0..80] OF CHAR;
```

PROCEDURE SaveCover(fn: ARRAY OF CHAR; VAR cv: cover);

This procedure, and those listed below, perform the operations indicated by the name on each data element comprising the cover. '**SaveCover**' performs '**SaveCoords**', '**SaveLineSet**', '**SaveChainList**' and '**SavePolygonSet**' on the appropriate data within the cover.

PROCEDURE LoadCover(fn: ARRAY OF CHAR; VAR cv: cover);

```
PROCEDURE InitCover(VAR cv: cover);

PROCEDURE EraseCover(VAR cv: cover);

PROCEDURE EmptyCover(VAR cv: cover);

PROCEDURE CloseCover(VAR cv: cover);

PROCEDURE GetPolygon(p: point; VAR cv: cover): polygon;

END covers.
```

• MODULE thesis;

This is the main program. Main programs do not have a definition and an implementation part. Some of the actual code is shown below to illustrate how relatively easy it is to put togther a main module once all the necessary procedures and functions are available in the respective service modules.

The following procedures are imported:-

```
FROM coord IMPORT WrCoord,FindCoord,AddCoord,-
RemoveCoord,CreateCoordSet,EraseCoordSet,point,coordset,-
WrCoordList,LoadCoords,SaveCoords,GetFirstCoord,GetNextCoord,
CHECK, MaxXY,FindNearestToPoint,-
FindNearestToLine,leftright,MinXY,jn,rectpol, sameside,-
intersect,EmptyCoordSet;

FROM surveyIO IMPORT Input,indev,outdev,buffer,commands,-
InputNo,InputInt,WrNo,getpoint,GetBearing,GetDistance,
print,WrDMS,WrSt,WrLn,WrCo,WrSp,logfile,graph;

FROM stackadt IMPORT definestack,erasestack,push,pop,-
stackempty,stack;

FROM queueadt IMPORT definequeue,erasequeue,addto-
queue,removefromqueue,queueempty,queue;

FROM lines IMPORT line,lineset,WrLine,WrLineList,SaveLines,-
LoadLines,CreateLineSet,EraseLineSet,AddLine,RemoveLine,-
FindLine,GetFirstLine,GetNextLine,LinesFrom,EmptyLineSet

FROM polygons IMPORT idlist,chain,chainlist,MakeNodeList,-
WrIdList,CreateIdList,EraseIdList,GetNextSide,GetFirstId,-
MakeChain,WrChain,MakeChainList,WrChainList,CreateChainList,
GetChain, ChainsFrom,GetFirstChain ,MakePolygons,-
ReverseChain,polygonset,ListPolygons,ListPolygon,polygon,-
FindPolygon,PolygonToString,ChainsToLineSet,EmptyChainList,
EmptyPolygonSet,InsidePolygon,WhichPolygon,AddLabel;
```

```
FROM Storage IMPORT HeapTotalAvail,MainHeap;

FROM covers IMPORT SaveCover,LoadCover,InitCover,
cover,EraseCover, EmptyCover,CloseCover,GetPolygon;

IMPORT InitJPI;
```

This module exports no procedures to the program. However because it is imported into the program its initialisation code is executed when the program starts up. The function of this code is to enable the 'Repertoire' modules to work with the JPI compiler.

```
FROM MATHLIB IMPORT ATan2,Sqrt,Cos,Sin,Tan;
```

The following modules are imported in their entirety. This is called a 'qualified' importation. All the procedures in the module are available to the main program but the procedure name must be qualified by the module name e.g. Str.Append(...

```
IMPORT Str;

IMPORT FIO;

IMPORT IO;

IMPORT Graph,grafix;


CONST rad = 57.29577951308;


VAR
    data      :cover;
    ls        :lineset;
    pt,x      : point;
    done      :BOOLEAN;
    fn        : ARRAY[0..80] OF CHAR;
    op        : CARDINAL;
    drawn     : BOOLEAN;
    px,py     : ARRAY[0..100] OF LONGREAL;
```

PROCEDURE enter;

This procedure takes a co-ordinate from the keyboard or a disc file according to the value of the input device variable '**indev**' and stores it in the coordset 'data.cs' (an indexed file)

```
BEGIN
    getpoint(pt);
    AddCoord(pt,data.cs);
    WrLn();
    WrSp(40);
    WrCo(pt);
    WrLn();
END enter;
```

PROCEDURE enterline;

This procedure shows how an objective can be attained by different means. When the screen is in the 'text' mode lines to be stored are read from the device set by 'indev' in text format. However in graphics mode the line connecting two points pointed to on the screen is added.

```
VAR
 p    : line;
 t    : ARRAY[0..10] OF CHAR;
 ok   : BOOLEAN;
 a,b,c : point;
BEGIN
  IF NOT graph THEN
    Input('Enter string - end with / : ',t);
    IF Str.Compare('/',t) = 0 THEN RETURN END;
    p.pt1: = CARDINAL(Str.StrToCard(t,10,ok));
    LOOP
```

```
Input('Enter string - end with / : ',t);
IF Str.Compare('/',t) = 0 THEN RETURN END;
p.pt2: = CARDINAL(Str.StrToCard(t,10,ok));
IF Str.Compare('/',t) = 0 THEN RETURN END;
    AddLine(p,data.ls);
    p.pt1: = p.pt2;
END;
ELSE
    REPEAT
        grafix.Move_Cursor;
        grafix.Get_World_Cursor(a.north,b.north);
        b: = FindNearestToPoint(a,data.cs);
```

Find the nearest point to the point where the cursor was place on the screen.

```
        p.pt1: = b.id;
        grafix.Move_Cursor;
        grafix.Get_World_Cursor(a.north,b.north);
        c: = FindNearestToPoint(a,data.cs);
        p.pt2: = b.id;
        AddLine(p,data.ls);
        grafix.DrawLine(b.north,b.east,c.north,c.east)
```

Draw the line selected.

```
    UNTIL grafix.button = 2;
END;
END enterline;
```

The next group of procedures do nothing except call procedures from the serrvice modules. The code could just have easily been written here in the main module but then it would not have been re-usable by other main modules.

```
PROCEDURE writelines;
BEGIN
    WrLineList(outdev,data.ls);
```

```
END writelines;

PROCEDURE loadline;
VAR
    fn: ARRAY[0..80] OF CHAR;
BEGIN
    Input('Line file : ',fn);
    LoadLines(fn,data.ls);
END loadline;

PROCEDURE deletecoord;
BEGIN
    pt.id:=InputInt('Point to delete ; ');
    RemoveCoord(pt,data.cs);
END deletecoord;

PROCEDURE listcoords;
BEGIN
    WrCoordList(outdev,data.cs);
END listcoords;

PROCEDURE savecoords;
BEGIN
    Input('File Name : ',fn);
    SaveCoords(fn,data.cs);
END savecoords;

PROCEDURE savecover;
BEGIN
    Input('File Name : ',fn);
```

```
        SaveCover(fn,data);
END savecover;

PROCEDURE loadcoords;
BEGIN
        Input('File Name : ',fn);
        LoadCoords(fn,data.cs);
END loadcoords;

PROCEDURE loadcover;
BEGIN
        Input('File Name : ',fn);
        LoadCover(fn,data);
END loadcover;

PROCEDURE showcoords;
```

Display the co-ordinate of a point on the screen.

```
BEGIN
        pt.id:=InputInt('Name to display : ');
        x:=FindCoord(pt.id,data.cs);
        WrSp(10) WrCo(x);
        WrLn();
END showcoords;

PROCEDURE endoff;
```

Close the files and return from the program to the operating system.

```
BEGIN
        CloseCover(data);
        done:=TRUE;
        FIO.Close(logfile);
```

END endoff;

PROCEDURE join;

PROCEDURE bisect(VAR r,s: LONGREAL);

PROCEDURE polar;

PROCEDURE chains;

PROCEDURE datasheet;

PROCEDURE Print;

The status of the global variable 'print' determines whether or not output data should be echoed to the printer.

PROCEDURE help;

PROCEDURE check;

THis procedure toggles the global variable 'CHECK' which determines whether or not the operator should be asked before over-writing a co-ordinate value.

PROCEDURE intersection;

PROCEDURE log;

This procedure opens a 'log' file to which all input data will be copied.

PROCEDURE inputfile;

Input will be switched from the keyboard to a named file. This might be a specially prepared batch file, or a 'log' file which would enable previous computations to be repeated.

PROCEDURE divideline;

Compute points on line.

PROCEDURE traverse;

The traverse routine is interesting from a programming point of view because it illustrates the use of an abstract data type known as a 'queue'. The queue is a 'first in first out' structure. This is ideal for a traverse calculation. As each data item (point, bearing, distance) is entered it is used to compute a preliminary value for the next point and placed in the queue. On reaching the end of the traverse the misclosure is computed. The data is now removed from the queue

in the order in which it was entered and the traverse in re-computed with the appropriate corrections made to the co-ordinates.

```
VAR
    a,b,c,d: point;
    pr : ARRAY[0..80] OF CHAR;
    totdist,jndist,r,s,r1,s1,fact,dx,dy,mdist,mbrg,cx,cy: LONGREAL;
    nos,names: queue;
    name: CARDINAL;
    open,hanging: BOOLEAN;
BEGIN
    pr:='Pt1 brg dist Pt2 brg dist Pt3 .... Ptn  [/ or Pt1] : ';
    Str.Append(pr,CHR(13));Str.Append(pr,CHR(10));
    a.id:=InputInt(pr);
    definequeue(nos,r);
    definequeue(names,name);
    totdist:=0.0;
    IF a.id0 THEN
        a:=FindCoord(a.id,data.cs);
        b:=a;
        addtoqueue(names,ADR(a.id));
            LOOP
            r:=GetBearing(data.cs);
            s:=GetDistance(data.cs);
            addtoqueue(nos,ADR(r));
            addtoqueue(nos,ADR(s));
            c.north:=a.north+Cos(r/rad)*s;
            c.east :=a.east +Sin(r/rad)*s;
            totdist:=totdist+s;
            name:=InputInt('Next Point : ');
            addtoqueue(names,ADR(name));
```

```
        IF buffer[0] = '/' THEN
            open: = TRUE;
            EXIT;
        END;
        IF name = b.id THEN
            open: = FALSE;
            EXIT;
        END;
        a: = c;
    END;
    IF open THEN
        d: = FindCoord(name,data.cs);
        IF d.id = 0 THEN
            hanging: = TRUE;
            dx: = 0.0; dy: = 0.0;
            mdist: = 0.0; mbrg: = 0.0;
            cx: = 0.0; cy: = 0.0;
        ELSE
            hanging: = FALSE;
            dx: = d.north-c.north;
            dy: = d.east-c.east;
            rectpol(dx,dy,mbrg,mdist);
            cx: = dx/totdist; cy: = dy/totdist;
        END;
    ELSE
        dx: = b.north-c.north;
        dy: = b.east-c.east;
        rectpol(dx,dy,mbrg,mdist);
        cx: = dx/totdist; cy: = dy/totdist;
    END;
```

```
            WrSp(40); WrCo(b); WrLn();
            removefromqueue(names,ADR(a.id));
            WHILE NOT queueempty(names) DO
                a:=FindCoord(a.id,data.cs);
                removefromqueue(nos,ADR(r));
                removefromqueue(nos,ADR(s));
                b.north:=a.north+Cos(r/rad)*s+cx*s;
                b.east :=a.east +Sin(r/rad)*s+cy*s;
                removefromqueue(names,ADR(b.id));
                IF (nameb.id) OR hanging THEN
                    AddCoord(b,data.cs);
                END;
                jn(a.id,b.id,data.cs,r1,s1);
                WrSt(' obs  '); WrDMS(r); WrNo(s,12,3); WrLn();
                WrSt(' adj  '); WrDMS(r1); WrNo(s1,12,3);WrSp(6);WrCo(b); W
                a:=b;
            END;
            IF NOT hanging THEN
                WrLn();WrSt('  Misclosure 1/'); WrNo(totdist/mdist,12,3);
                WrSp(10); WrNo(dx,12,3); WrNo(dy,12,3);
            END;
        END;
        erasequeue(names);
        erasequeue(nos);
        WrLn();
END traverse;

PROCEDURE memfree;
PROCEDURE setextent(VAR cs: coordset);
```

Set the limits of the area to be displayed on the graphic screen.

PROCEDURE PlotPoints(VAR cs: coordset);

PROCEDURE PlotLines(VAR x: lineset; VAR c: coordset);

PROCEDURE DrawLabels(VAR ps: polygonset);

PROCEDURE plot;

Set extent to include the all the points in the cuurent cover, the plot points, lines, and labels.

PROCEDURE zoom;

Change the area displayed on the screen.

PROCEDURE listpolys;

Print data sheet for all polygons within the current cover.

PROCEDURE listpoly;

Print the data sheet for a particular polygon.

```
VAR
    p: polygon;
    A: LONGREAL;
BEGIN
    p.id:=InputInt('Polygon No.: ');
    FindPolygon(p.id,p,data.ps);
    A:=ListPolygon(outdev,p,data.ch,data.cs);
END listpoly;
```

PROCEDURE build;

This procedure forms a chainlist from a lineset and then creates a polygonset from the chains. Finally the data for all the polygons is listed.

```
VAR
    nl: idlist;
BEGIN
    CreateIdList(nl);
    MakeChainList(data.ls,data.ch);
```

```
    MakePolygons(data.ps,data.ch,data.cs);
    ListPolygons(outdev,data.ps,data.ch,data.cs);
    EraseIdList(nl);
END build;
```

PROCEDURE PickPolygon(VAR p:polygon);

A procedure to select a polygon on the graphic screen. The polygon selected is returned in variable 'p'.

```
VAR
 X,Y : LONGREAL;
    q  : point;
BEGIN
    grafix.Move_Cursor;
    grafix.Get_World_Cursor(X,Y);
    q.north:=Y;
    q.east:=X;
    p:=GetPolygon(q,data);
END PickPolygon;
```

PROCEDURE label;

This procedure gets a label point from the the cursor on the graphic screen, determines which polygon the point is in, then gets a label from the keyboard and attaches the values to the polygon record.

```
VAR
    X,Y : LONGREAL;
    q : point;
    p : polygon;
    l : ARRAY[0..8] OF CHAR;
BEGIN
```

```
    IF NOT graph THEN WrSt('Use command DISPLAY first'); WrLn;
RETURN END;
    grafix.Move_Cursor;
    grafix.Get_World_Cursor(X,Y);
    q.north:=Y; q.east:=X;
    p:=GetPolygon(q,data);
    p.labx:=q.north;
    p.laby:=q.east;
    Input('Label : ',p.label);
    AddLabel(p,p.label,p.labx,p.laby,data.ps);
    grafix.DrawText(q.east,q.north,0,p.label);
END label;

PROCEDURE clear;
BEGIN
    EraseCover(data);
END clear;

PROCEDURE cls;
```

PROCEDURE textscr;

Set screen to text mode.

PROCEDURE graphscr;

Set screen to graphics mode.

PROCEDURE set;

Used to set graphic attributes such as foreground and background colours and line types.

PROCEDURE GetCommand(): CARDINAL;

This procedure takes a command from the keyboard and returns a numeric value corresponding to the position of the command in the command list.

The main part of the program follows:-

```
BEGIN
```

Initialisation:

```
    indev:=FIO.StandardInput;

    outdev:=FIO.StandardOutput;

    graph:=FALSE;

    drawn:=FALSE;

    buffer:='';

    CHECK:=TRUE;

    commands:='ENTER DELETE ? LIST SAVE LOAD EXIT JOIN POLAR
DATA PRINT HELP CHECK ';

    Str.Append(commands,"INTER LOG INPUT TRAVERSE LINE CHAIN
PLOT TEST FREE ");

    Str.Append(commands,"ENTERLINE BUILD LOADLINE LISTLINE
LOADCOORDS LISTPOLYS ");

    Str.Append(commands,"LISTPOLY CLEAR CLS TEXT DISPLAY
ZOOM SET LABEL ");

    done:=FALSE;

    InitCover(data);

    grafix.OpenWorkStation(grafix.CGA);

    ChainsToLineSet(data.ch,data.ls);
```

The main loop:-

```
    REPEAT

        op:=GetCommand();

        CASE op OF

        0:  enter;|

        1: deletecoord;|

        2: showcoords;|

        3: listcoords;|

        4: savecover;|

        5: loadcover;|

        6: endoff;|
```

```
            7: join;|
            8: polar;|
            9: datasheet;|
            10: Print;|
11: help;|
12: check;|
13: intersection;|
14: log;|
15: inputfile;|
 16: traverse;|
17: divideline;|
18: chains;|
19: plot|
20: test;|
21: memfree;|
22: enterline;
23: build;|
24: loadline;|
25: writelines;|
26: loadcoords;|
27: listpolys;|
      28: listpoly;|
      29: clear;|
      30: cls;|
      31: textscr;|
      32: graphscr;|
33: zoom;|
34: set;|
35: label;
 END;
```

UNTIL done;

'done' is set to **TRUE** by the procedure '**endoff**' which closes all the open files.

END thesis.

• MODULE ModiStru;

This program is used to modify the structure of the data files to create fields to hold attribute data.

The first step is to determine the structure of the existing files making up the cover. The existing fields are then displayed to the user who is prompted to enter the new fields required. If an existing field (other than the fields determining the topological structure of the data) is omitted it will not be included in the new files.

The next step is to create empty files with the new structure and to copy data from the old files to the new for fields with the same name. The final step is to delete the old cover and rename the new files to the same name as the old ones.

```
FROM covers IMPORT cover,InitCover,CoverName,CloseCover,
        EraseCover,RenameCover,CreateCover;
FROM NdxFiles IMPORT FileStructure,CreateNdxFile,GetField,
        PutField, WriteRecord,GetFieldAdr,PutFieldAdr;
FROM NdxUtils IMPORT GetRecName;
FROM surveyIO IMPORT Input;
IMPORT FIO,IO,Str,InitJPI;

TYPE
fl = ARRAY[0..200] OF CHAR;
rec = ARRAY[0.. 19] OF CHAR;
```

```
PROCEDURE EraseC(old: ARRAY OF CHAR);
VAR             po : ARRAY[0..80] OF CHAR;
BEGIN
    Str.Copy(po,old);
    Str.Append(po,'.cxo');
    FIO.Erase(po);
    Str.Copy(po,old);
    Str.Append(po,'.cxn');
    FIO.Erase(po);
    Str.Copy(po,old);
    Str.Append(po,'.pxl');
    FIO.Erase(po);
END EraseC;

VAR
    flcs,flch,flps : fl;
    nflcs,nflch,nflps : fl;
    oflcs,oflch,oflps : fl;
    d1,d2 : cover;
    rn: rec;
    h1,h2,h3,i,j,code,sz1: CARDINAL;
    add1: ADDRESS;
    OK: BOOLEAN;
    of : rec;
    OldCover: fl;

BEGIN
    CoverName:="";
    IO.ClrScr;
```

```
InitCover(d1);
h1:=d1.cs^.handle;
h2:=d1.ch^.handle;
h3:=d1.ps^.handle;
FileStructure(d1.cs,flcs);
FileStructure(d1.ch,flch);
FileStructure(d1.ps,flps);
IO.WrStr(flcs);IO.WrLn;
IO.WrStr(flch);IO.WrLn;
IO.WrStr(flps);IO.WrLn;
Input('New Field List for Points   : ',nflcs);
Input('New Field List for Chains   : ',nflch);
Input('New Field List for Polygons : ',nflps);
Str.Caps(nflcs);
Str.Caps(nflch);
Str.Caps(nflps);
(* Check for compulsory fields *)
IF Str.Pos(nflcs,'POINT')=MAX(CARDINAL) THEN
    Str.Concat(nflcs,'POINT,',nflcs)
END;
IF Str.Pos(nflch,'POINTS')=MAX(CARDINAL) THEN
    Str.Concat(nflch,'POINTS,',nflch)
END;
IF Str.Pos(nflch,'RIGHT')=MAX(CARDINAL) THEN
    Str.Concat(nflch,'RIGHT,',nflch)
END;
IF Str.Pos(nflch,'LEFT')=MAX(CARDINAL) THEN
    Str.Concat(nflch,'LEFT,',nflch)
END;
IF Str.Pos(nflch,'END')=MAX(CARDINAL) THEN
```

```
        Str.Concat(nflch,'END,',nflch)
    END;
    IF Str.Pos(nflch,'START') = MAX(CARDINAL) THEN
        Str.Concat(nflch,'START,',nflch)
    END;
    IF Str.Pos(nflps,'LABY') = MAX(CARDINAL) THEN
        Str.Concat(nflps,'LABY,',nflps)
    END;
    IF Str.Pos(nflps,'LABX') = MAX(CARDINAL) THEN
        Str.Concat(nflps,'LABX,',nflps)
    END;
    IF Str.Pos(nflps,'LABEL') = MAX(CARDINAL) THEN
        Str.Concat(nflps,'LABEL,',nflps)
    END;
    IF Str.Pos(nflps,'INSIDE') = MAX(CARDINAL) THEN
        Str.Concat(nflps,'INSIDE,',nflps)
    END;
    IF Str.Pos(nflps,'SIDES') = MAX(CARDINAL) THEN
        Str.Concat(nflps,'SIDES,',nflps)
    END;
```

Make a list of existing fields which go into new file

```
    i:=0;
    oflcs:='';
    oflch:='';
    oflps:='';
    LOOP
        Str.ItemS(of,flcs,',',i);
        IF Str.Compare(of,'') = 0 THEN EXIT END;
        IF Str.Pos(nflcs,of)AX(CARDINAL) THEN
            Str.Append(oflcs,of);
```

```
            Str.Append(oflcs,",");
        END;
        INC(i);
    END;
    i:=0;
    LOOP
        Str.ItemS(of,flch,',',i);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        IF Str.Pos(nflch,of)AX(CARDINAL) THEN
            Str.Append(oflch,of);
            Str.Append(oflch,",");
        END;
        INC(i);
    END;
    i:=0;
    LOOP
        Str.ItemS(of,flps,',',i);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        IF Str.Pos(nflps,of)AX(CARDINAL) THEN
            Str.Append(oflps,of);
            Str.Append(oflps,",");
        END;
        INC(i);
    END;
    i:=0;
    IO.WrLn;
    IO.WrStr(oflcs);IO.WrLn;
    IO.WrStr(oflch);IO.WrLn;
    IO.WrStr(oflps);IO.WrLn;
    IO.WrLn;
```

```
IO.WrStr(nflcs);IO.WrLn;
IO.WrStr(nflch);IO.WrLn;
IO.WrStr(nflps);IO.WrLn;
Str.Copy(OldCover,CoverName);
```

Create new files with a temporary name with the new file structure.

```
CreateNdxFile(d2.cs,'$t.cxo',nflcs);
CreateNdxFile(d2.ch,'$t.cxn',nflch);
CreateNdxFile(d2.ps,'$t.pxl',nflps);
```

Copy Existing Fields

```
i:=1;
WHILE GetRecName(d1.cs,i,rn) DO
    j:=0;
    LOOP
        Str.ItemS(of,oflcs,',',j);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        OK:=GetFieldAdr(d1.cs,rn,of,code,add1,sz1);
        OK:=PutFieldAdr(d2.cs,rn,of,code,add1,sz1);
        INC(j);
    END;
    OK:=WriteRecord(d2.cs,rn);
    INC(i);
END;
i:=1;
WHILE GetRecName(d1.ch,i,rn) DO
    j:=0;
    LOOP
        Str.ItemS(of,oflch,',',j);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        OK:=GetFieldAdr(d1.ch,rn,of,code,add1,sz1);
        OK:=PutFieldAdr(d2.ch,rn,of,code,add1,sz1);
```

```
            INC(j);
        END;
        OK:=WriteRecord(d2.ch,rn);
        INC(i);
    END;
    i:=1;
    WHILE GetRecName(d1.ps,i,rn) DO
        j:=0;
        LOOP
            Str.ItemS(of,oflps,',',j);
            IF Str.Compare(of,'')=0 THEN EXIT END;
            OK:=GetFieldAdr(d1.ps,rn,of,code,add1,sz1);
            OK:=PutFieldAdr(d2.ps,rn,of,code,add1,sz1);
            INC(j);
        END;
        OK:=WriteRecord(d2.ps,rn);
        INC(i);
    END;
```

Close the files comprising the new cover, erase the old, and re-name the new.

```
    FIO.Close(h1);
    FIO.Close(h2);
    FIO.Close(h3);
    CloseCover(d2);
    EraseC(OldCover);
    RenameCover('$T',OldCover);
END ModiStru.
```

• MODULE de;

A very simple data entry program to add attribute data to graphic elements.

```
FROM covers IMPORT cover,InitCover,CoverName,CloseCover;

FROM NdxFiles IMPORT FileStructure,CreateNdxFile,GetField,-
PutField,WriteRecord,GetFieldAdr,PutFieldAdr;

FROM NdxUtils IMPORT GetRecName;

FROM surveyIO IMPORT Input,outdev,InputInt, WrSt, WrLn, WrCo,
WrNo;

IMPORT IO,Str,InitJPI,Lib;

FROM coord IMPORT point;

FROM polygons IMPORT idlist,WrIdList;

FROM GenLists IMPORT ListDelete,ListLength,StrCode;


TYPE
    fl = ARRAY[0..200] OF CHAR;
    rec = ARRAY[0.. 19] OF CHAR;


VAR
    flcs,flch,flps : fl;
    nflcs,nflch,nflps : fl;
    oflcs,oflch,oflps : fl;
    d1 : cover;
    rn,sel: rec;
    i,j,code,sz1: CARDINAL;
    add1: ADDRESS;
    OK: BOOLEAN;
    of : rec;


PROCEDURE GetPointAttributes;
VAR
```

```
    i: CARDINAL;
    of,og: rec;
    ok: BOOLEAN;
BEGIN
    i:=InputInt('Point No.: ');
    Str.CardToStr(LONGCARD(i),rn,10,ok);
    j:=0;
    LOOP
        Str.ItemS(of,flcs,',',j);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        Str.Copy(og,of);
        OK:=GetFieldAdr(d1.cs,rn,of,code,add1,sz1);
        IF Str.Compare(of,'POINT')=0 THEN
            WrCo(point(add1^));WrLn;
        ELSE
            Str.Append(of,'          ');
            of[10]:=CHR(0);
            Lib.Move(add1,ADR(nflcs),sz1);
            nflcs[sz1]:=CHR(0);
            WrSt(of); WrSt(' : ');WrSt(nflcs);WrLn;
            Input('New Data : ',oflcs);
            IF Str.Length(oflcs)0 THEN
                OK:=PutField(d1.cs,rn,og,StrCode,oflcs);
                OK:=WriteRecord(d1.cs,rn);
            END;
        END;
        INC(j);
    END;
    INC(i);
END GetPointAttributes;
```

```
PROCEDURE GetChainAttributes;
VAR
    i: CARDINAL;
    of,og: rec;
    ok: BOOLEAN;
    ls: idlist;
BEGIN
    i:=InputInt('Chain No.: ');
    Str.CardToStr(LONGCARD(i),rn,10,ok);
    j:=0;
    LOOP
        Str.ItemS(of,flch,',',j);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        OK:=GetFieldAdr(d1.ch,rn,of,code,add1,sz1);
        IF NOT OK THEN EXIT END;
        Str.Copy(og,of);
        Str.Append(og,'          ');
        og[10]:=CHR(0);
        Lib.Move(add1,ADR(nflcs),sz1);
        nflcs[sz1]:=CHR(0);
        WrSt(og); WrSt(' : ');
        IF Str.Pos('START END LEFT RIGHT',of)AX(CARDINAL) THEN
            IO.WrCard(CARDINAL(add1^),5);WrLn;
        ELSIF Str.Compare(of,'POINTS')=0 THEN
            OK:=GetField(d1.ch,rn,of,code,ls);
            WrIdList(outdev,ls);
        ELSE
            WrSt(nflcs);WrLn;
            Input('New Data : ',oflcs);
```

```
            IF Str.Length(oflcs)0 THEN
                OK:=PutField(d1.ch,rn,of,StrCode,oflcs);
                OK:=WriteRecord(d1.ch,rn);
            END;
        END;
        INC(j);
    END;
    INC(i);
END GetChainAttributes;

PROCEDURE GetPolygonAttributes;
VAR
    i: CARDINAL;
    of,og: rec;
    ok: BOOLEAN;
    ls: idlist;
    lx: LONGREAL;
BEGIN
    i:=InputInt('Polygon No.: ');
    Str.CardToStr(LONGCARD(i),rn,10,ok);
    j:=0;
    LOOP
        Str.ItemS(of,flps,',',j);
        IF Str.Compare(of,'')=0 THEN EXIT END;
        OK:=GetFieldAdr(d1.ps,rn,of,code,add1,sz1);
        IF NOT OK THEN EXIT END;
        Str.Copy(og,of);
        Str.Append(og,'          ');
        og[10]:=CHR(0);
        Lib.Move(add1,ADR(nflcs),sz1);
```

```
            nflcs[sz1]:=CHR(0);
            WrSt(og); WrSt(' : ');
            IF Str.Pos('ID ',of)AX(CARDINAL) THEN
                IO.WrCard(CARDINAL(add1^),5);WrLn;
            ELSIF Str.Pos('LABX LABY',of)AX(CARDINAL) THEN
                OK:=GetField(d1.ps,rn,of,code,lx);
                WrNo(lx,10,3);WrLn;
            ELSIF Str.Pos('SIDES INSIDE',of)AX(CARDINAL) THEN
                OK:=GetField(d1.ps,rn,of,code,ls);
                IF OK THEN
                    WrIdList(outdev,ls);(*WrLn;*)
                END;
                ListDelete(ls,1,ListLength(ls));
            ELSE
                WrSt(nflcs);WrLn;
                Input('New Data : ',oflcs);
                IF Str.Length(oflcs)0 THEN
                    OK:=PutField(d1.ps,rn,of,StrCode,oflcs);
                    OK:=WriteRecord(d1.ps,rn);
                END;
            END;
            INC(j);
        END;
        INC(i);
    END GetPolygonAttributes;

    BEGIN
        CoverName:='';
        IO.ClrScr;
```

Open the cover and read the file structure

```
InitCover(d1);
FileStructure(d1.cs,flcs);
FileStructure(d1.ch,flch);
FileStructure(d1.ps,flps);
```

Display the file strucutre

```
IO.WrStr(flcs);IO.WrLn;
IO.WrStr(flch);IO.WrLn;
IO.WrStr(flps);IO.WrLn;
LOOP
    sel:="";
    REPEAT
        Input('(P)oint, (C)hain, p(O)lygon attribute, E to Exit (P,C,O or
',sel);
        WrLn;
        Str.Caps(sel);
    UNTIL sel[0] IN IO.CHARSET{'P','C','O','E'};
    IF sel[0]='P' THEN GetPointAttributes END;
    IF sel[0]='C' THEN GetChainAttributes END;
    IF sel[0]='O' THEN GetPolygonAttributes END;
    IF sel[0]='E' THEN EXIT END;
END;
CloseCover(d1);
END de.
```

Appendix B

Functional Specifications for a Land Information System

Introduction

Below is given an outline of the functions which are necessary or desirable in an 'open' Land Information Sytem which has the function of

1) storing and managing all 'sharable' data from a number of participating agencies,

2) serving as a central hub for the exchange of data between these agencies, and

3) performing spatial analysis and statistical computations with regard to spatial entities and their associated attributes or non-spatial data.

DATABASE

Integrated Database

The database is responsible for storage, retrieval, evaluation, analysis and data management of graphic and non-graphic information. The graphic and non-graphic data must appear to the user as an integrated database. The other main considerations for this database are that

duplication in input, storage, processing of data and structure must be minimised

there must be no contradiction between graphic and non-graphic data

data integrity must be ensured

Non-Graphic Data

The database must store attribute data of any graphic object or of any non-graphical record which has no link to a graphic object.

Multiple Data Types Support

The database must support a variety of data types including single precision integers and floating point numbers, double precision integers and real numbers, character strings and dates.

Query Language

The database must support a powerful database query language to access non-graphic data either by interactively identifying graphic objects or by search criteria.

Schematic Drawings

The database must have the ability to handle both schematic drawings and scale maps in the same database, in such a way that similar graphic entities in both the schematic and scale maps can share common non-graphic data.

One Map

The integrated database must cover the whole of the territory of the participating agencies in one continuous map. Storage, update, and access to any portion of the continuous map must be done without the need for the user to supply any map numbers or map boundaries. It must be possible to create sub-layers of the continuous map to meet the requirements of different applications or tasks.

Performance Maintained with Large Data Set

The database must be able to handle large amounts of multi-layered, heterogeneous, and spatially indexed data without substantial degradation of retrieval and analysis performance.

Topological Structure

Graphical entities must be topologically structured.

Security

The system must provide control of access to data elements to limit read/write access to specially authorised users. It must also be possible to control 'read only' access to certain classes of confidential data. Backup and recovery features must be available.

To protect the integrity of the master database, any update must be done by extracting the update area into a workfile in which editing is done. After checking and authorisation, this workfile is used to update the master database. When a workfile is extracted from the master database with the

intention of carrying out an update, the corresponding graphic and non-graphic data in the master database should be available to other users for 'read-only' access, and users should be notified via a system prompt that the data in question is being updated.

Data Transfer

The database must be able to exchange data and link up with other external databases.

DATA ENTRY

Attribute Data Capture

The system must allow attribute data to be entered into the database from

a) alpha-numeric terminal

b) bulk loading from pre-formatted ASCII files

c) graphics workstation

It is important that input from a) and c) uses a customised form input environment which performs all data integrity checking and database operations for inserting data and establishing relationships in the database.

Customised Data Entry Procedures

Data capture procedures must be customised for easy-to-learn, efficient use.

Survey Field Data

The system should be capable of supporting a customised interface for the interactive input of field data from survey field books or electronic data recorders.

The system must allow the user to review and edit the input data and provide formatted output capability for data verification.

A survey computation process is required to compute and adjust the survey data and co-ordinate the positions of surveyed features. The system should then have the capability to generate a plot from the survey and create any necessary links to attribute data in the database.

The system must be able to carry out updating based on such survey data without the need to specify map boundaries.

Digital Graphic Exchange

The system must support digital graphic exchange with particular emphasis on industrial standard exchange formats, and especially those supporting topological structures.

It must also be capable of being interfaced to a digital stereoplotter for on-line photogrammetric mapping.

Co-ordinate Geometry

The system must support a co-ordinate geometry software which allows for the entry of survey data from field notes or digital files. The software should be capable of operating on data created using co-ordinate geometry routines to perform civil engineering and surveying functions.

Input from Multiple Sources

The system must allow data to be digitized from a number of maps covering a common geographic area. These maps may vary in scale, map projection and accuracy. The system must also allow digitized data to be automatically transformed during digitization so that it is interactively merged with the existing spatial database. The database must be automatically updated without regard for map sheets or map boundaries (continuous map).

Validity Checks on Data Entry

The system must be capable of performing validity checks on data as they are entered. These procedures should include checks for valid format, range, and checks against a list of values.

Co-ordinate Transformations

The system must include procedures to transform digitized points from digitizer co-ordinates to the national grid.

The system must be able to display co-ordinate values representing cursor or mouse location on the digitizer.

The system should also be able to carry out a least squares adjustment on the control points which are used for the map digitizing set-up. The magnitude and direction of these control point residuals must be displayed for analysis. Such a feature allows the operator to accept or reject the digitizer set-up or any other similar transformation operation.

The system must provide a choice of transformations for the digitizer set-up including affine, Helmert, and projective transformations.

Automatic Linework Cleaning and Error Identification

The system must support linework cleaning operations such that.

a) small gaps in lines can be automatically closed.

b) overshoots at line intersections can be eliminated resulting in an intersection point on the line intersected.

c)small undershoots at line intersections can be automatically closed resulting in an intersection point being created on the line intersected.

The application software must be capable of automatically flagging all those digitizing errors that cannot be corrected by the automatic linework cleaning operations, and drawing the operator's attention to them. This feature allows the operator to correct errors interactively.

Elimination of Double Points or Lines

The system must support automatic reduction of unnecessary co-ordinate detail through line generalisation using user-defined tolerances.

Polygon Formation

The system must be capable of automatically forming polygons from clean linework. Each polygon must be assigned an identifier by the system of a user-defined pattern.

Automatic Attribute Creation

The system must be capable of automatically creating attributes for point, line, and polygon features such as ID-numbers, polygon areas and perimeters, and line lengths. These attributes should be stored in the database.

Automatic Map Update

The system must provide automatic map updating techniques such as polygon update ("cut and paste"), polygon elimination ("sliver removal"), polygon dissolve (dropping lines between polygons with the same attributes), attribute code change, splitting arcs, thinning and weeding arc co-ordinates, and matching nodes.

Topological Consistency Checking

The system must be capable of converting files containing points and lines (e.g.ASCII SIF-format files) into a topological structure. During the conver-

sion, checking for topological consistency must be performed automatically. Examples of such checks are

a) Does every line segment join two nodes?

b) Is every polygon bounded properly by line segments?

c) Are there any holes (areas not covered by polygons)?

d) Are there any overlays (areas covered by more than one polygon)?

DATA EDITING

The applications software must allow operators to interactively edit data that has been entered. These functions allow users to correct any errors or omissions detected during the entry of spatial and attribute data. These functions must include the following capabilities:-

Interactive Editing

The system must support on-line interactive editing of data that has been entered such as to add to or modify existing graphic entities and their attributes. Editing functions must include, but no be limited to: copying, moving, deleting, undeleting, modifying and adding graphic entities.

The system must permit users to interactively edit any point and line data, within a polygon, and within a selected range of "layers", including

a) user defined patterning and rotation of pattern symbols

b) user defined symbols

c) squaring buildings

d) alignment of buildings to linear features (such as roads) by interactive editing.

e) manuscript preparation with sheet grid and grid labeling

f) labeling lines with bearings and distances

g) working in an azimuthal co-ordinate system rather than plain Cartesian

h) area calculation of closed elements

i) parallel line functions (such as copy a line or curve parallel to another at a given offset)

j) display of digitized features by line mode and symbol mode

k) adjustment of traverse

l) extend line at a given distance

m) extend two lines to their intersection

n) erase part of a graphic feature

o) erase all features to one side of a given line

Group Manipulations

The system must allow operators to temporarily associate a group of graphic elements so that a single command can operate on all the elements in the group at one time. Group placement functions must include place group, block or shape, modify or move group, and support the capability to manipulate the elements contained within a group individually. The software should also support fence contents manipulations such as copy, delete, move, scale, rotate, change symbology, and change colour/thickness, pattern of the group of elements.

Edge Matching

The system must support the capability to automatically match graphic data across artificial boudaries such as map sheets and photogrammetric models. Edge matching would be transparent to the user.

Data Update

The system must allow users to interactively update data that has been entered. These types of functions must include the following capabilities:

Selective Map Updating Capability

The system must allow areas of maps to be erased and/or updated with new data. Updated areas should be extracted from the master database into a workfile in which all editing would be done. When completed and authorised the workfile is updated to the master database.

Independent Attribute Data Updating

The system must have the capability to update attribute data independently from or in conjucntion with cartographic data.

DATA MANAGEMENT

The system should be capable of storing and managing map features and their associated attributes. The attribute data management operations must be based on a relational database system with the following capabilities:

Data Dictionary

The system must incorporate a data dictionary defining file content and format.

Map Annotation

It must support point, line, polygon, and map annotation features.

Graphic and Attribute Linkage

An automatic linkage mechanism must be provided for attaching graphic and non-graphic attributes so that it appears to the user an an integrated graphic and non-graphic database.

Map Modification

The user must be allowed to copy, rename, append or delete any portion of the continuous map by specifying map boundaries.

Dynamic Redefinition of the Database

The database must provide for dynamic re-definition of database content and format (e.g. add new fields, delete or redefine existing fields etc.)

Map Extraction

Map extraction functions must allow users to access and extract information from the spatial database without compromising the integrity of the database. Extraction operations should allow the user to :

1) specify a polygon co-ordinate window, spatial sub-division, or user-specified geographic area to be extracted (e.g. a circle with a specified radius)

2) specify a layer(s) to be extracted

3) clip each layer of interest to the specified co-ordinate window

4) join layers across spatial subdivisions to form a full map

5) copy the extracted data into a user workspace for further use.

Map Insertion

The map insertion function must allow users to do

1) Add maps to the database, automatically partitioning them according to the defined spatial and thematic (layer) structure.

2) Insert maps without regard to size, area covered, or scale.

3) verify that all the maps have been correctly added.

DATA MANIPULATION, ANALYSIS AND QUERY

All data manipulation and analysis procedures must operate on the cartographic and attribute components of the database. Users should be able to operate on each component separately or treat them as an integrated unit. All data modifications necessary because of these operations must be handled automatically by the software.

The following spatial analysis functions are necessary:

Point, line, and polygon overlay

Overlay points on polygons

Overlay lines on polygons to create new line features

Overlay two sets of polygons to create a new set of polygon features with attributes from both input maps.

Reclassification and aggregation of attribute data.

Geometric operations such as rotation, translation and transformation of co-ordinates to specific projections.

Overlay functions must permit Boolean 'AND' 'OR' and 'XOR' operations.

Analysis functions must generate 'clean' topological results in the same format as the input maps.

Areas and perimeters of new polygons, lengths of new lines, and intersection co-ordinates of old and new lines must be computed automatically.

Map Manipulation

Provide capabilities for co-ordinate transformations between various map projections and co-ordinate systems in both interactive and batch mode. The software must also support data conversion between different geodetic datums.

The system should be able to generate aggregates or subsets of maps by selecting map features with specific attributes of interest.

Attribute Manipulation

Users should be able to manipulate attributes independently from the cartographic features associated with them. The software should include the following attribute manipulation functions:-

Selectively retrieve records based on a Boolean combination of attributes (e.g. AND, OR, XOR, EQ, NE, GE, LT, LE). It should also be possible to build composite Boolean expressions.

Perform rapid search for polygons or other graphic features within a specified search area given user-defined attribute criteria.

Perform rapid search for non-graphic data by interactively identifying graphic entities via selection criteria.

It is desirable that the system perform the following manipulations:-

1. Calculate the value of new or existing attribute fields using arithmetic operations which should include +, -, *, /.

2. Automatically aggregate a set of polygons on the basis of a common attribute or attributes.

3. Perform adjacency analysis on a set of polygon data.

Interactive Query

The software should allow the user to query any part of the data that has been created. Query operations should support the following:-

1. facilities for user-friendly query capability from both graphic and alpha-numeric environments.

2. selection of map features by attribute, location, or by selection criteria consisting of Boolean or arithmetic expressions.

3. searches for map features by location, attribute, or combination of attributes.

4. queries generated in the graphics environment must be able to access the attribute database and present the results of the query in a graphical manner e.g. by highlighting features or by colour filling.

5. storing the results of queries for later recall and display.

6. search operation within a restricted area in order to enhance performance.

NETWORK ANALYSIS

Network Data Management Requirement

Network data management software should consist of a set of operations which create, analyse, and display networks such as city streets, canals, railways and utility lines. The operations should allow users to simulate operations which occur in real networks such as allocating, routeing or distributing resources, facilities, or services throughout a network based on movement restrictions such as time, distance, resource supply and demand for resource use.

The software should be capable of capturing, storing and analysing topologically correct networks. The network software must be built from the structure of the basic software rather than as a separate system.

Network Database Requirements

The database requirements for network data management are similar to those for the basic software. In addition, requirements that are specific to network database management must allow users to :

1) Use the same features and topological data structures as the basic software.

2) directly interface with both the cartographic and attribute handling components of the basic software

3) operate on files containing attributes defining directional impedance interrelation between characteristics of each network links and nodes

4) operate on files containing attributes defining impedance for the intersection of any two particular network links

5) operate on files containing line attributes defining an unlimited number of street or other names for each network link

6) support the use of length indicators such as mileposts.

Data Entry Requirements

Data entry capablilty of the software should allow users to:

1) create topological networks consisting of links and nodes using the basic software

2) automatically edit and update networks using all relevant capabilities of the basic software.

Network Analysis Requirement

Analysis capabilities of the software should allow users to use any network data with the operations in the basic software. In addition users should be able to:

Appendix C

Systems in the Cape Town Municipality Containing GIS Data

1. Landuse

2. Trade Wastes System

3. Effluent Billing System

4. Census

5. Building File

6. Property Purchases

7. Plans Tracking System

8. Amenities System

9. Street Tree Inventory System

10. Tuberculosis Record and Control System

11. Birth Register

12. Death Register

13. Street Register

14. Town Survey Marks

15. Survey Filing System

16. Tacheometer System

17. Departure Application System

18. Mechanical Ventilation System

19. Substation Maintenance System

20. City Land Register

21. Bridges System

22. Mitchells Plain Housing System

23. Rezoning Application System

24. Pavement Management System

25. Manhole System

APPENDIX D

Syntax Structure of Modula-2

Compilation Unit

```
\
Program Module
|\
| "MODULE"Ident[Priority]";"{Import}Block Ident"."
|          \
|          "["ConstExpression"]"
|
|
Implememtation Module
|\
| ""IMPLEMENTATION" "MODULE"Ident[Priority]";"{Import}Block Ident"."
|
|
Definition Module
\
 "DEFINITION" "MODULE" Ident";"{Import}{Definition}"END" Ident"."
```

```
Import
\
["FROM" Ident] "IMPORT" IdentList";"
```

```
Block
\
 {Declaration}["BEGIN" StatementSequence]"END"
 \
 "CONST"{ConstantDeclaration";"}
 |    \
 |     Ident"="ConstExpression
 |
 "TYPE"{TypeDeclaration";"}
 |  \
 |   Ident"="Type
 |
 "VAR"{VariableDeclaration";"}
 |  \
 |   IdentList":"Type
 |
 ProcedureDeclaration";"
 |
 Module Declaration";"



Definition
\
 "CONST"{ConstantDeclaration";"}
 |    \
 |     Ident"="ConstExpression
 |
 "TYPE"{TypeDeclaration";"}
 |  \
 |   Ident"="Type
 |
 "VAR"{VariableDeclaration";"}
 |  \
 |   IdentList":"Type
 |
 ProcedureHeading";"
```

```
ProcedureDeclaration
\
 ProcedureHeading";"Block Ident
 \
  "PROCEDURE" Ident[FormalParameters]
              \
               "("{FPDefinition/";"}")"[":"QualIdent]
                 \
                  ["VAR"]IdentList":"FormalType
                              \
                               ["ARRAY" "OF"]QualIdent


ModuleDeclaration
\
 "MODULE" Ident[Priority]";"


ConstExpression
\
 SimpleConstExpression[Relation SimpleConstExpression]
 \
  ["+"|"-"] ConstTerm { AddOperator ConstTerm }
         \
          ConstFactor {MulOperator ConstFactor }
          \
           Number
           |
           String
           |
           QuaaIIdent
           |
           ConstSet
           |\
           | [QualIdent]"{" ConstElementList "}"
           |             \
           |              { ConstElement/"," }
           |                \
```

```
        |           ConstExpression [".."ConstExpression]
        |
        "(" ConstExpression ")"
        |
        NegationOperator ConstFactor


Statement
\
EmptyStatement
|\
|
AssignmentStatement
|\
| Designator ":=" Expression
|
ProcedureCall
|\
| Designator[ ActualParameters ]
|         \
|          "(" ExpressionList ")"
|
IfStaement
|\
| "IF" Expression "THEN" StatementSequence {ElsifPart} [ElsePart] "END"
|                              |       \
|                              |        "ELSE" StatementSequence
|                              \
|                               "ELSIF" Expression "THEN" StatementSequence
|
CaseStatement
|\
| "CASE" Expression "OF" {Case/"|"} [ElsePart] "END"
|                 \
|                  [CaseLabelList ":" StatementSequence]
|
WhileStatement
|\
| "WHILE" Expression "DO" StatementSequence "END"
```

```
|
ForStatement
|\
| "FOR" ControlSection "DO" StatementSequence "END"
|    \
|      Ident ":=" Expression "TO" Expression ["BY" ConstExpression]
|
RepeatStatement
|\
| "REPEAT" StatementSequence "UNTIL" Expression
|
LoopStatement
|\
| "LOOP" StatementSequence "END"
|
WithStatement
|\
| "WITH" Designator "DO" Statement Sequence "END"
|
ReturnStatement
|\
| "RETURN" Expression
|
ExitStatement
\
 "EXIT"


Type
\
QualIdent
|
SimpleType
|\
| EnumerationType
| |\
| | "(" IdentList ")"
| |
| SubrangeType
```

```
| \
|  [QualIdent] "[" ConstExpression ".." ConstExpression "]"
|
StructuredType
|\
| SetType
| |\
| | "SET" "OF" SimpleType
| |
| ArrayType
| |\
| | "ARRAY" SimpleType{ "," SimpleType} "OF" Type
| |
| RecordType
| \
|  "RECORD" FieldListSequence "END"
|        \
|        {FieldList/";"}
|         \
|          IdentList ":" Type
|          |
|          UnionType
|
PointerType
|\
| "POINTER" "TO" Type
|
ProcedureType
\
 "PROCEDURE" [FormalTypeList]
          \
          "(" {FormalSpecification/ ","} ")" [":" QualIdent]
             \
              ["VAR"] FormalType

UnionType
\
 "CASE" [Ident]":" QualIdent "OF" {Variant/"|"}["ELSE" FieldListSequence] "END"
                    \
```

CaseLabelList ":" FieldListSequence

```
Expression
\
 SimpleExpression [Relation SimpleExpression]
\            \
 |            "=" | "<>" | "<" | "<=" | ">" | ">=" | # | "IN"
 |
 ["+" |  "-"] Term {AddOperator Term}
          \    \
           |    "+" | "-" | "OR"
           |
          Factor { MulOperator Factor }
          \      \
           |      "*" | "/" | "DIV" | "MOD" | "AND" | "&"
           |
          Number
           |
          String
           |
          Variable
           |\
           | Designator
           |
          Set
           |\
           | [QualIdent] "{" ElementList "}"
           |       \
           |        {Element/ ","}
           |         \
           |          Expression [".." ConstExpression]
           |
          FunctionCall
           |\
           | Designator [ActualParameters]
           |
          "(" Expression ")"
           |
          NegationOperator Factor
```

```
        \
        "NOT" | "~"

QualIdent
\
Ident {"." Ident}

Designator
\
QualIdent {Selector}
        \
        "." Ident
        |
        "[" ExpressionList "]"
        |
        "^"


CaseLabelList
\
CaseLabels {"," CaseLabels}
\
ConstExpression [".." ConstExpression]


IdentList
\
{Ident/","}


ExpressionList
\
{Expression/","}


StatementSequence
\
{Statement/";"}
```