Cemac Tate access floors help Queensland TAB stay ahead of the field.

The new TAB computer centre in Brisbane controls an on-line, real time communications network which links 104 off-course TAB offices in the Brisbane and Gold Coast areas, a telephone betting centre and 52 country centres.

Cemac Tate Access Floors along with Cemac partitions were installed in both computer and general office areas, which means that future expansion or changes can be made without disruption of TAB services to punters.

Cemac Tate's all metal system offers four panel sizes. You can specify the unique 'levelock' base which ensures a perfectly level floor even on a sloping slab. The panels have 176 weldments to provide all the strength you need, and you can choose from a wide range of finishes including long lasting carpet, wood, cork, or vinyl asbestos tiles.

Cemac Interiors offers single source responsibility for complete fitting out of office interiors including floors, ceilings, partitions and task lighting.

So if you are looking for an office building that gives you significant operational savings and infinite flexibility when you plan or change work areas, then like TAB Queensland, you'll be backing a winner with Cemac Tate Access Floors.

Brochures and details from Cemac Interiors
SYDNEY: ............................ 6993122
MELBOURNE: ..................... 637811
BRISBANE: ......................... 2215099
CANBERRA: ....................... 823706
ADELAIDE: ......................... 453656
PERTH: ............................ 247888
HOBART: ........................... 344582

CID0004 x FPM
The
AUSTRALIAN COMPUTER
JOURNAL

Volume 9

March, 1977
to
November, 1977
# Contents

<table>
<thead>
<tr>
<th>AUTHOR/S</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.S. Northcote</td>
<td>A More Effective ACS?</td>
<td>4-6</td>
</tr>
<tr>
<td>J.M. Bennett</td>
<td>The Australian Computer Society</td>
<td>7-8</td>
</tr>
<tr>
<td>G. Karoly</td>
<td>Real-Time Systems at the Victorian TAB — An Historical Review</td>
<td>11-16</td>
</tr>
<tr>
<td>D.M. Deighton</td>
<td>Computers in Operations Research</td>
<td>17-20</td>
</tr>
<tr>
<td>C.S. Wallace</td>
<td>Computing Research in Australia</td>
<td>21-24</td>
</tr>
<tr>
<td>A.W. Goldsworthy</td>
<td>Computers and Their Social Implications in the Australian Scene</td>
<td>25-31</td>
</tr>
<tr>
<td>W.N. Holmes</td>
<td>The Major Tasks of Data Processing</td>
<td>32-38</td>
</tr>
<tr>
<td>M.W. Barry</td>
<td>Computing in the Printing Industry</td>
<td>39-41</td>
</tr>
<tr>
<td>J.L. Keedy</td>
<td>An Outline of the ICL 2900 Series System Architecture</td>
<td>53-62</td>
</tr>
<tr>
<td>A.H.J. Sale</td>
<td>Primitive Data Types</td>
<td>63-71</td>
</tr>
<tr>
<td>J.R. Penny</td>
<td>A Simple Model for Input-Output and Computer Overlap</td>
<td>72-76</td>
</tr>
<tr>
<td>J.D. Day</td>
<td>Stability of Serial Hybrid Computer Solution of Linear Parabolic Partial Differential Equations</td>
<td>77-79</td>
</tr>
<tr>
<td>A.D. Payne</td>
<td>Error Control for Computer Communication</td>
<td>80-87</td>
</tr>
<tr>
<td>B. Dwyer and K. Hutchings</td>
<td>Flowchart Optimisation in COPE a Multi-Choice Decision Table</td>
<td>92-100</td>
</tr>
<tr>
<td>G.K. Gupta</td>
<td>A Course About Social Implications of Computers</td>
<td>101-106</td>
</tr>
<tr>
<td>B.J. Austin</td>
<td>Job Scheduling by a Front End Computer</td>
<td>107-111</td>
</tr>
<tr>
<td>J.P. Higgins</td>
<td>A Computer Program for Exploring the Homogeneity of Randomly Sampled Categorical Data</td>
<td>112-113</td>
</tr>
<tr>
<td>J.A. Self</td>
<td>Artificial Intelligence in Computer-Assisted Instruction</td>
<td>118-127</td>
</tr>
<tr>
<td>G.N. Lance and W.T. Williams</td>
<td>Attribute Contributions to a Classification (Short Communication)</td>
<td>128-129</td>
</tr>
<tr>
<td>C.M. Prime</td>
<td>An Integrated System for the Administration of Managed Funds</td>
<td>132-137</td>
</tr>
<tr>
<td>H. Mackenzie and J.L. Smith</td>
<td>The Implementation of a Database Management System</td>
<td>138-144</td>
</tr>
<tr>
<td>B.G. Cook and K.K. Mayo</td>
<td>A Geographic Data Base Using FORDATA</td>
<td>145-149</td>
</tr>
<tr>
<td>J.D. Shortridge</td>
<td>Development of a Bibliographic Search System Using FORDATA</td>
<td>150-154</td>
</tr>
<tr>
<td>H. Mackenzie and G. Kelly</td>
<td>A Query/Update Package for Library or Personal Reference Use</td>
<td>155-158</td>
</tr>
<tr>
<td>H.W. Holdaway</td>
<td>GINV, a Subroutine in ANSI FORTRAN for Generalized Matrix Inversion</td>
<td>159-161</td>
</tr>
<tr>
<td>D.M. Boulton</td>
<td>High Density Digital Recording</td>
<td>162-168</td>
</tr>
<tr>
<td>Editorial</td>
<td></td>
<td>3, 52</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td>6, 38, 44, 49, 71, 87-89, 100, 106, 111, 129, 144, 154</td>
</tr>
<tr>
<td></td>
<td>Tenth Anniversary Edition 1966-1976</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>The Roll of Honour. The Australian Computer Society</td>
<td>9-10</td>
</tr>
<tr>
<td></td>
<td>Invitations to Contributors to submit short communications to “The Australian Computer Journal”</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>ACS 8 Call for Proposals</td>
<td>42-44</td>
</tr>
<tr>
<td></td>
<td>Notes on Submission of Papers</td>
<td>45-48</td>
</tr>
<tr>
<td></td>
<td>Notes for Guidance of Referees</td>
<td>48-49</td>
</tr>
<tr>
<td></td>
<td>Forthcoming National and International Events</td>
<td>50, 130, 169</td>
</tr>
</tbody>
</table>
Index for the Australian Computer Journal, Vol. 9

1. Index of Titles of Papers, Books Reviewed, etc.

A Computer Program for Exploring the Homogeneity of Randomly Sampled Categorical Data. 112
A Course About Social Implications of Computers. 101
ACS 8 Call for Proposals. 42
Advances in Computers Vol. 14. 71
Advances in Computers Vol. 15. 111
A Geographic Data Base using FORDATA. 145
A More Effective ACS? 4
Analytic Computational Complexity. 100
An Integrated System for the Administration of Managed Funds. 132
An Outline of the ICL 3900 Series System Architecture. 53
A Query/Update Package for Library or Personal Reference Use. 155
Artificial Intelligence Techniques in Computer Assisted Instruction. 118
A Simple Model for Input-Output and Computer Overlap. 72
Attribute Contributions to a Classification. 128
Batch Submission, Demand Terminal and "Hands-on" Computing in a Computer Science Tertiary Education Environment. 114
Computational Methods and Problems in Aeronautical Fluid Dynamics. 87
Computers and the Future of Education. 88
Computers and their Social Implications in the Australian Scene. 25
Computers in Operations Research. 17
Computing in the Printing Industry. 39
Computing Research in Australia. 21
Development of a Bibliographic Search System using FORDATA. 150
Error Control for Computer Communication. 80
Flowchart Optimisation in COPE on Multi-Choice Decision Table. 92
Forthcoming National and International Events. 50, 130, 169
GINV, A Subroutine in ANSI FORTRAN for Generalized Matrix Inversion. 159
Handbook of Data Communications. 49
High Density Digital Recording. 162
Introduction to PASCAL. 154
Invitations to Contributors to Submit Short Communications to "The Australian Computer Journal". 31
Job Scheduling by a Front End Computer. 107
Multi-Variable Control Theory. 144
Notes for Guidance of Referees. 48
Notes on Submission of Papers. 45
Physics of Computer Memory Devices. 144
Primitive Data Types. 63
Real-Time Systems at the Victorian TAB - An Historical Review. 11
Review of "Data Base Directions - The Next Steps". 38
Review of "Selection and Acquisition of Data Base Management Systems". 44
Review of "Studies in Operation Systems". 44
Revised Report on the Algorithmic Language ALGOL 68. 6
RTI/2 Design Philosophy. 154
Stability of Serial Hybrid Computer Solution of Linear Parabolic Partial Differential Equations. 77
Tenth Anniversary Edition 1966-1976. 2
The Australian Computer Society. 7
The Implementation of a Database. 138
The Major Tasks of Data Processing. 32
The Present and Future Use of Computers in Australia and Employment Implications. 88

2. Index to Authors and Other Contributors

Austin, B.J. 107
Barry, M.W. 39
Bennett, J.M. 7
Boulton, D.M. 162
Cook, B.G. 145
Creasy, P.N. 114
Dabke, F.P. (BR). 144
Day, J.D. 77
Deighton, D.M. 17, (BR). 88
Dwyer, B. 92
Foo, N.T. (BR). 100
Frith, D.A. (BR). 87
Goldsworthy, A.W. 25
Gupta, G.K. 101
Heale, K. (BR). 144
Heron, D.L. (BR). 49
Higgins, J.F. 112
Holdaway, H.W. 159
Holmes, W.N. 32
Hutchings, K. 92
Jarvis, R.A. 114
Juliff, P. (BR). 88
Karoly, G. 11
Kelly, G. 155
Lance, G.N. (SC) 128
McKee, H. 138, 155
Mayo, K.K. 145
Northcote, R.S. 4
Payne, A.D. 80
Penny, J.R. 72
Prime, C.M. 132
Sale, A.H.J. 63, (BR). 71
Sanderson, J.G. (BR). 111
Self, J.A. 118
Shortridge, J.D. 150
Smith, J.L. 138
Topor, R.W. (BR). 6
Wallace, C.S. 21
Williams, W.T. (SC). 128
Woodhouse, D. (BR). 88

3. Computing Reviews Categories Index

1. GENERAL TOPICS AND EDUCATION
   1.2 History, Biographies. 7, 11, 17, 21, 25, 32
   1.3 Introductory and Survey Articles. 7, 17, 32, 39
   1.5 Education
      1.50 General. 118
      1.52 University Courses and Programs. 101, 114
   2. COMPUTING MILIEU
      2.0 General. 101
      2.1 Philosophical and Social Implications. 32
      2.11 Economic and Sociological Effects. 25
      2.12 The Public and Computers. 25
      2.2 Professional Aspects. 4, 21
      2.4 Administration of Computing Centres
         2.44 Equipment Evaluation. 72
   3. APPLICATIONS
      3.1 Natural Sciences.
3.12 Biology. 112
3.14 Earth Sciences. 145
3.3 Social and Behavioral Sciences. 112
3.32 Education; Welfare. 118
3.4 Humanities
3.43 Literature. 39
3.5 Management Data Processing
3.50 General. 92
3.51 Education; Research. 150
3.52 Financial. 132
3.59 Miscellaneous. 39
3.6 Artificial Intelligence
3.60 General. 118
3.7 Information Retrieval
3.70 General. 132, 138, 155
3.73 File Maintenance. 132, 138, 145
3.74 Searching. 132, 150
3.8 Real-Time Systems
3.80 General. 11, 107
3.81 Communications. 11, 80

4. SOFTWARE
4.1 Processors
4.19 Miscellaneous. 92
4.2 Programming Languages
4.20 General. 63, 114
4.22 Procedure- and Problem-Oriented Languages. 92, 138, 155
4.3 Supervisory Systems
4.30 General. 114
4.32 Multiprocessing; Multiprocessing. 53
4.33 Data Base. 138, 145
4.34 Data Structures. 145
4.35 Operating Systems. 107
4.4 Utility Programs
4.49 Miscellaneous. 92
4.9 Miscellaneous. 63

5. MATHEMATICS OF COMPUTATION
5.1 Numerical Analysis
5.14 Linear Algebra. 159
5.17 Ordinary and Partial Differential Equations. 77
5.2 Metatheory
5.24 Analysis of Programs; schemata; semantics; correctness. 63, 92
5.4 Mathematical Programming. 128
5.5 Mathematical Statistics; Probability. 112, 128
5.6 Information Theory. 80

6. HARDWARE
6.2 Computer Systems
6.20 General. 72
6.21 General-Purpose Computers. 53
6.3 Components and Circuits
6.35 Input/Output Equipment. 162
6.4 Patents, Hardware. 162

8. FUNCTIONS
8.1 Simulation and Modelling. 72
8.3 Operations Research/Decision Tables. 17, 92
**NEW COMPUTERISED SECURITY ALARM SYSTEM**

A new computerised monitoring and processing system is poised to revolutionise the Australian security industry.

Mr. Bob Stagg demonstrates the new Coral computer security system, which embodies the use of a mini-computer and VDU and teleprinter, together with data terminal equipment. The demonstration was given to more than 100 business firms' representatives in NSW recently.

The compact system enables a single guard to monitor and control over 14,000 potential security risks and malfunctions — from fires, explosions and bank hold-ups to changes in room temperature — while seated in front of a video display terminal.

The computer, through a system of switches and sensors, can detect a vast range of events, warn the guard with a tone signal then flash detailed information about each incident on to the video screen.

The system is being marketed by Sydney-based security and fire specialists, Fire Fighting Enterprises (Aust.) Ltd (FFE).

As well as warning of each specific threat or breakdown, the unique system gives the guard a list of instructions to combat it.

If a series of events occur simultaneously, the computerised system makes a split-second evaluation of their priority and any possible links, then gives the guard his course of action.

Routine events, such as a drop in water pressure in a building's sprinkler system, can be automatically corrected by the computer.

The system, named CORAL, makes the widely-used “panel light” warning method obsolete.

American tests have shown the human error factor rises sharply if a single operator has more than 150 panel warning lights to monitor.

The CORAL system eliminates decision making for the guard and greatly reduces the human error factor.

A major advantage of the CORAL system is that the guard doesn’t have to be a computer expert to function efficiently — he merely follows the instructions which appear on the video screen.

A serial printer attached to the computer keeps a permanent record of messages which appear on the visual display unit. Every event which occurs during a shift is recorded by the printer, along with the guard’s comments.

The system was developed in America by the Walter Kidde Corp. and utilises computer hardware manufactured by Digital Equipment.

The nerve centre of the CORAL system is compact and consists of a PDPIV03 mini-processor, a VT50 video terminal and an LA36 serial printer, all of which can be housed in an area the size of an executive office desk.

The central computer is linked to the monitored areas by 14 telephone lines. Each telephone line is capable of supporting 64 Data Collection Panels which in turn can each monitor 16 different events.

The CORAL system is equipped with several failsafe mechanisms, including instant alert if a telephone line is cut, emergency power supply and a series of changeable codes to prevent unauthorised access to the computer.

The computer can send up to 56 commands, either automatic or manual, through each telephone line.

**DATA GENERAL ANNOUNCES VERSATILE NEW LANGUAGE**

A high-level structured programming language that allows users to develop programs for a wide range of products from a microcomputer to a high-end minicomputer has been announced by Data General Australia Pty. Ltd.

The new language, DG/L, is aimed at software product OEMs, in-house data processing service groups, and university computer centres.

DG/L’s program transportability allows compilation, without source change, for execution on Eclipse, Nova, and microNova computers under the multiprogramming Advanced Operating System (AOS), Real-time Disc Operating System (RDOS), diskette-based Disc Operating System (DOS), or Real-Time Operating System (RTOS).

“The DG/L language and its concepts require technically qualified and sophisticated computer users,” said Mr. Wayne Fitzsimmons, General Manager, Data General Australia. “These users are highly system software-oriented, compared to the many minicomputer users whose software needs fall into the applications development area.”

At the systems programming level, DG/L’s direct operating system interface permits complex utility development without assembly language programming. Further, the multi-tasking run-time environment of DG/L makes it particularly useful for multi-terminal system development under RDOS and RTOS.

Because DG/L is derived from ALGOL, it is ideally suited for structured programming techniques, where small
blocks of code with specific functions are organised into a tree structure based on the logical flow of the program. This modular arrangement of functions makes the program easier to read, debug and maintain.

DG/L's source language syntax lets a programmer code in one form regardless of the type of Data General system that will run the programs. For example, within an AOS Program development system, code can be written, compiled and debugged under AOS and passed as an executable file to smaller systems. Code generation alternatives include AOS for ECLIPSE, and RDOS, DOS or RTOS for ECLIPSE and NOVA-line computers.

INTRODUCING SYSTEM/34 — A SMALL WORK-STATION COMPUTER FROM IBM

IBM Australia have announced the IBM System/34, a computer with visual displays and printers, which make it well-suited to the work-station method of working.

The IBM System/34 can handle data from its origin through to the printed invoice, statement, stock report and statistical analysis. It can be 'tailored' to fit in with a company's working methods.

A small company might use System/34 for all its data processing needs. A current IBM System/370 or large System/3 user might use System/34 as a local processing system, with a teleprocessing link to the host computer.

The system consists of a system unit with 32K, 48K or 64K bytes of main storage. It has disk storage of 8.6, 13.2 or 27.1 million bytes. And a diskette unit for such purposes as entry of data and data security. A diskette can hold up to 303,104 bytes of data, or with a double-sided, double-density diskette, 1,212,416 bytes.

To the system unit, the user can attach a 'mix' of up to eight newly-designed IBM 5251 Display Stations and IBM 5256 Printers. Each display station can be up to 1500 metres from the system unit, except for the one which serves as the system console, which has to be within 6 metres of, and in the same room as the system unit.

System/34 is designed to run several programs concurrently. The IBM System Support Program Product controlling the system enables different users to work on different jobs at the same time, each effectively independent of the others.

First European customer shipment of System/34 is planned for February, 1978.

NEW BWD MINI-LAB

A new version of the BWD 6038 Mini-Lab has now been released by BWD Electronics Pty Ltd.

The BWD 6038 Mini-Lab is a remarkable instrument, combining no less than five power supplies, two amplifiers, a wide range function generator and ramp generator. Whilst primarily designed for education, it is equally applicable to both industry and research.

In one compact cabinet it provides a function generator; range of 0.001Hz to 2MHz; sine, square or triangle with variable mark space offset and AM or FM modulation and sweep ramp generator.

The Mini-Lab conforms closely to I.E.C. 348 safety recommendations and the power transformer to I.E.C. 76, section 35, 36, 37 and 40. It has no projecting heat sinks, has detachable power cord and Switch selection for 110 or 230V AC operation.

By retaining a similar panel layout, the BWD 603B can be used adjacent to a BWD 603A without confusion or the need to re-write experiments.

Further information is available from BWD Electronics Pty Ltd, Mile Street, Mulgrave, Victoria.

HEWLETT-PACKARD OFFERS NEW POWERFUL DESKTOP COMPUTER SYSTEM

The first in a new generation of desktop computer systems has been placed on the market by Hewlett-Packard Company.

The new Series 9800 System 45 has the most powerful central processor and the largest built-in mass storage system ever offered in a desktop computer. It also features a 12-inch CRT display, BASIC interpretive language conforming to the new ANSI standard, applications software, and an optional graphics package with high-speed hard-copy output. The system is all contained within a single compact package.

The Series 9800 System 45 is 18.5 in. high, 19 in. wide and 26¾ in. deep, and weighs 75 lbs (34.1 kg), making...
it well-suited for applications requiring portability, such as use in different areas within a large company. The basic system, with built-in keyboard, 16 kbytes of read/write memory, CRT and one tape transport, costs about $15,000, plus tax. Delivery is 16 weeks.

DRANETZ MODEL 606

Power Line Disturbance Analyser

You know all about power line behaviour and your installation!

The Dranetz Model 606 Line Disturbance Analyser will provide data to enable you to pinpoint the effects of irregular line quality of your equipment.

CONSIDER THESE FUNCTIONS:

- Complete round-the-clock record (up to 99 days) of power line disturbances and interruptions in permanent printed format.
- Monitors, classifies and records time, type and magnitude of each disturbance. Time clock may be set to real or system time.
- Switch selectable threshold values for impulse (Duration 0.5 & 100 μ secs.). Cycle to cycle changes in RMS levels and slow changes in steady state RMS levels.
- Internal computer accumulates statistical data for daily summary readout. Data available any time on command.
- Internal batteries provide 4 hours operation during power down time.

NOW CONSIDER THESE FEATURES:

- Isolated inputs for 115/230/460 V RMS. Single and three phase models.
- Operates only on disturbance or command—no wasted paper.
- Immediate printout of line disturbances.
- Excellent for permanent monitoring.
- Portable (mass 6.8 kg), weatherproof case.
- Easy to connect, easy to read, permanent record for analysis.

For further details contact:

ELECTRICAL EQUIPMENT LTD
86 Liverpool Street, Sydney, N.S.W. Telephone 61 8321
8 Remont Court, Cheltenham, VICTORIA. Telephone 550 4522
25 Maud Street, Newstead, QUEENSLAND. Telephone 52 3962
why these well-known companies...

- Qantas — Qantas, Sydney
- Bank of N.S.W., Sydney, Brisbane and Perth
- Savings Bank of S.A., Adelaide
- I.B.M. Sydney and Melbourne
- Ansett, Melbourne
- Commercial Banking Co. of Sydney
- Aust. National University, Canberra
- C.A.G.A., Sydney
- T.A.B. of N.S.W., Sydney
- Government Printing Office, Canberra

choose Rintoul

INFINITE ACCESS FLOORS

Full protection for all cables, piping and ducting yet fully accessible for maintenance, service or modification, that's Rintoul Infinite Access Floors. Rintoul offer two types of infinite access floors... discuss your needs with out technical representative.

- Sole Australian Agents: Floating Floors Inc., U.S.A. — Computer Floors
- Airflow Company, U.S.A. — Air Conditioning Units.

Suppliers of INFINITE ACCESS FLOORING
- OFFICE PARTITIONS AND SCREENS
- PLANTER BOXES
- FOLDING DOORS
- DETAIL JOINERY
- ALL CLASSES OF VENEER WORK
- Ask for our colour brochure.

Rintoul E.D.P. Services
A DIVISION OF RINTOUL PTY LIMITED
26 Powers Road, (P.O. Box 226) Seven Hills, N.S.W. 2147 Tel.: (02) 624 5333
Sales Office: 10 Minogue Crescent, Glebe, N.S.W. 2037 Telephone: (02) 660 6677
Qld.: R.P.L. Pty. Ltd., 20 Duncan Street, (P.O. Box 80) Fortitude Valley 4006
Tel.: 52 8338, 52 4849, 52 4663.
Contents:

132-137  An Integrated System for Administration of Managed Funds
By C. PRIME

138-144  The Implementation of a Data Base Management System
by H.G. MACKENZIE and J.L. SMITH

145-149  A Geographic Data Base Using FORDATA
By G.C. COOK and K.K. MAYO

150-154  Development of a Bibliographic Search System Using FORDATA
By J.D. SHORTRIDGE

155-158  A Query/Update Package for Library or Personal Use
By H. MACKENZIE and G. KELLY

159-161  GINV, A Subroutine in ANSI Fortran for Generalized Matrix Inversion
By H.W. HOLDAWAY

162-168  High Density Digital Recording on an Audio Recorder
By D.M. BOULTON

Special Features:

144, 154  Book Reviews

169  Forthcoming National and International Events

170  IFIP Congress 80

REPRINTS: 50 copies of reprints will be provided to authors. Additional reprints can be obtained, according to the scale of charges supplied by the publishers with proofs. Reprints of individual papers may be purchased for 50 cents each from the Printers (Publicity Press).

PAPERS: Papers should be submitted to the Editor; authors should consult the notes published in Volume 9, pp. 45-49 (or request a copy from the National Secretariat).

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous in local matters, and may charge different membership subscriptions. Information may be obtained from the following Branch Honorary Secretaries. Canberra: P.O. Box 446, Canberra City, A.C.T. 2601. N.S.W.: P.O. Box N250, Grosvenor St, Sydney, N.S.W. 2000. Qld: Box 1484, P.O., Brisbane, Qld. 4001. S.A.: Box 2423, G.P.O., Adelaide, S.A. 5001. W.A.: Box F320, G.P.O., Perth, W.A. 6001. Vic.: P.O. Box 98, East Melbourne, Vic. 3002. Tas.: P.O. Box 216, Sandy Bay, Tas. 7005.
An Integrated System for Administration of Managed Funds

By C. Prime

This paper describes the design of a form of integrated system for a growing section of life office business.

Previous piecemeal attempts to computerize this section of business had been made, and with the passing of time their operation was becoming more and more unwieldy.

The query was frequently raised 'would it not be easier to administer such business manually?' At the same time it was clear that the manual work required was fast becoming prohibitive. One of the main problems with the older systems was seen to be lack of flexibility.

Consequently, the new system as described in the paper, was designed aiming to introduce every possible flexibility.

KEY WORDS AND PHRASES: Integrated system, master file, string of trailers, validation, file maintenance, output analysis.

C.R. CATEGORIES: 3.52, 3.70, 3.73, 3.74.

Introduction

The system described has been designed to cope with the administration problems of a minor but fast growing section of the business of most life assurance offices — referred to subsequently as 'managed funds'. The system as described below has been in use since 1972 by the Mutual Life and Citizens Assurance Co. Ltd. Since writing this paper it has been considerably extended mainly in the area of automatically generated accounting entries. The hardware configuration available has remained much the same and consists of:

- IBM 370/158
- 4 x 3420 tape drives
- 1 x 1403 printer
- 1 x 2540 card read punch
- limited space on 2314 disk drives sufficient only for work areas.

A managed fund is usually a vehicle for implementing a superannuation arrangement between an employer and his employees. It offers a flexibility of benefit structure which is becoming widely appreciated. Managed fund business written by life offices is increasing rapidly, and without mechanisation, the clerical work involved is likely to become overwhelming.

The essential feature of a managed fund is that the trustee takes out a policy with the life office, governed by a set of rules as to what shall be contributed by all parties and what benefits shall be received in return. The beneficiaries, referred to subsequently as 'members' are the people to be superannuated. The important feature of the membership is that it is transitory and covers different people at different times according to a set of rules, usually the employees of a firm at any point in time.

This type of business has usually been left out of any system designed to mechanize clerical work, because of the great variation of clerical procedures from one fund to another.

The system used is a special adaption of what is called an “integrated system”. The essence of an “integrated” system is that all relevant files are collected into one large file. One pass only is made against this file during the running of the system. Many different types of output may be obtained during this one pass, and those required from any particular run are selected and indicated to the system.

The basis of this system is a master file containing all the information about each policy (since life office business is referred to in terms of policies — every sale is a policy. The after sales service for managed funds is considerable). This file is complete and up-to-date. One program only operates on it. Alterations to it and requests for information from it (collectively called transactions) are effected by feeding the code numbers for such alterations and requests into this program. Other input is minimal because the file is complete.

The effort involved in setting up such a system is considerable because it anticipates so much of the day to day work of clerks and machine operators. Having made this initial effort, subsequent operation and development are correspondingly simplified.

1. THE FILES

In an integrated system everything revolves around the master file. It is a "super" file in all respects. In this case the master file is divided into two, the fund file and the member file.

These files are, as they must be in such a system, a complete record for each fund and each member. This eliminates the need for maintaining subsidiary files relating to funds or members and enables a request for processing to be made with a minimum of information supplied with the request and a consequent reduction in manual work and errors.

There are several common types of fund. The information required to be kept for each type varies considerably. A flexible system of record design is therefore necessary to minimise the size of the fund record and at the same time preserve the advantages of a standard layout.
To cater for the variations in information required for the different funds, a system of trailers is used. The record for a fund is actually a string of fixed length records (trailers) which all contain the policy number to identify them as belonging to the fund, and a “trailer number” to identify the type of trailer. The member file is similarly organised.

The trailers may be added and removed at will. Information is grouped into trailers as convenient, considering when and why it is required. Some trailers relate to unusual circumstances and are present only in a minority of cases. Other trailers are required only at particular times of the year and will later be deleted.

1.1 The Fund File

The fund file is a sequential tape file in policy number order. The types of trailer in use, are, very broadly:

(i) Standard Identification trailer: with a common format for all funds.
(ii) Subdivision trailers: also in a common format. One trailer for each membership subdivision within a fund.
(iii) Special purpose trailers: which are present only when and if required. A list of these trailers would never be final as new types of trailer are part of the development of the system.

1.2 The Member File

This is a sequential tape file in order of member name and date of birth within policy number.

There does not seem to be any practical alternative to keeping the members of a fund in order of name and date of birth. Alloting members numbers would only introduce complications. It would be unreasonable to expect the employer to include member numbers in all his correspondence and it would be more difficult to find the correct member when a wrong number is given than if the name is misspelt. Further, it is necessary to allow reference to the member by name alone for most requests.

However, this does pose the problem of change of key. By “key” is meant the information which determines the order of the file; in this case policy number/name/date of birth, and, in the case of the fund file, policy number.

Unlike policy number, which can be relied on to be correct, the name/date of birth information for a new member is frequently incorrect and needs to be changed later, resulting in a change to his key and consequently his position in the file. This is a big problem as far as the system is concerned. The technique used to solve it is described in 3. At this point it is appropriate to dismiss what may seem to be the obvious solution.

The initial appearance was that the member file needed to be accessed both sequentially and directly.

Sequential access is the facility to read the records one after the other without supplying the keys. This is necessary as it must be possible, given only the policy number, to access the records for all members of the fund. Direct access is the facility to find the record for a particular member, given his key, by going directly to it rather than starting at the beginning of the file and searching for it. This type of access only becomes necessary when a “change of key” is being processed. When the request for change of name/date or birth is detected, file reading will have reached the correct position in the file for the member. The incorrect record must then be accessed directly so that it can be written into correct position in the file.

An “indexed” file would seem to fit the bill. This type of file is kept on a direct access device, usually disk, and programming support for it is available in the software for most large commercial installations. It is basically a sequential file, but has an index so that direct reference may be made to a record when the key is supplied.

This, the software manual will boast, allows both sequential and direct access. So it does, but to have them both at the same time, as is essential in this case, involves a double description of the file in the program coding. The core storage requirements for this type of file are large enough without doubling them. More generally, “indexed” files seem to be very delicate flowers. They do not stand the rough treatment a programmer can get away with using sequential files.

Since it fails the “best method” test here, in what seems like an ideal application for it, the indexed file must be suspected of being very limited use, and of no use at all in an integrated system.

Again for members there are

(i) Standard Identification trailer
(ii) Special purpose trailers.

Records for members who have left the scheme are retained on the file and indicated to be out of force. It is envisaged that these records will be removed after a certain time lapse. New members (also new fund and members for new funds) may be set up and marked “not yet in force” as soon as details are available for them. In this way the file is complete, and benefit and contribution quotations for new members can be included in the system.

1.3 General

These files contain all data relevant to the funds and members. There is little information by way of history. Further trailers containing historical information could be added if required.

2. THE SYSTEM

The flow diagram shows one major difference from the normal flow pattern for an integrated system. The master files are read in “validation” as well as in “file maintenance”. It is thus a two pass system.

The features which made this desirable are:

(a) The interdependence of the correctness of output for a fund on the correctness of input for each member.
(b) The files are small and the volume of transactions low compared to, say, a system for all ordinary policies.

Fortunately (b) makes it possible to deal with the far reaching consequences of (a).

The output required from the system is frequently in the form of a list giving certain details for each member of a fund. Such a list is of no use unless the input details for each member are correct. A list with one member omitted because of an error in his input would be confusing.

The problem can be overcome by holding up the production of such a list until all the input needed for it has been checked. This can only be done by reading the master files at the validating stage before any part of the list has been generated.

For a huge file there might not be enough hours in the day to read the master file twice, completely, in every run. However, feature (b) makes 2 passes feasible in this case, and this should be so for some years even if business grows at a rapid rate.
Integrated System for Managed Funds

Transaction Cards

Validation Error List

Transaction Holding File

Transaction File

Fund Master File

Transaction Holding File

New Generation of Fund Master File

Member Master File

New Generation of Member Master File

New Generation of Member Master File

Output Messages

List of Transactions Processed and Transactions held on Holding File

Any other Output Requested

OUTPUT ANALYSIS

SORT

FILE MAINTENANCE

VALIDATION

INPUT TO NEXT RUN OF VALIDATION

The Australian Computer Journal, Vol. 9, No. 4, November, 1977
2.1 Validation

Validation is designed to detect all possible errors in the input and to transform the card images into a transaction file suitable for processing in file maintenance.

Batch editing is catered for by an optional batch total card at the end of each batch of input cards. The number of cards is checked. The total of any accountable amounts in the batch may be checked by punching the total into the same columns of the batch total card as the accountable amount appears in the detail cards.

Some errors cannot be detected without reference to the master files; for instance:
- a non-existent (but not invalid) policy number or member name/date of birth,
- a transaction requesting calculation which cannot be performed on the type of fund in question,
- a request to process a member’s withdrawal when he has already withdrawn.

Normally in an integrated system, such errors would not be detected until file maintenance processing, at which point the transaction would be cancelled. Here it would often be too late to cancel the transaction. Consider a run where two transactions for the same fund were requested, one transaction to change the date or birth of a member who happened to be the last member for the fund because his name began with Z, and another transaction to prepare a list of all members of the fund. Suppose Z’s name was incorrectly punched. By the time this was detected the list details would have been prepared for every other member and the appropriate output analysis messages set up. Cancellation of the change date of birth transaction would result in an incorrect list being automatically produced in “output analysis”.

Reading the master files in “validation” enables the error to be detected at this stage, so that all correct transactions for the fund may be “held” until the next run.

Errors are graded into two types. They are either significant or they may be ignored and an assumption made as to what was intended. The classification is made for each error according to which will cause the least trouble in the long run. Of the three errors given as examples above the first two are considered significant and the third is ignored, the assumption being that the transaction repeats a previous request for the Member’s withdrawal.

Validation stores all correct transactions on the transaction file. The transactions are not released for processing in file maintenance until a specific request is made to release them for a fund and then only if there are no significant errors in the input for that fund. The correct transactions are held and recycled via the transaction holding file to the next run until they are released.

This technique of holding transactions until the operator is satisfied there are no errors prevents partially correct lists being produced. It does cause some problems. At times it is necessary to know what transactions are on the transaction holding file.

2.2 File Maintenance

File maintenance deals with transactions released for processing in fund and member order, while the master files are being read and re-written. The backbone of file maintenance is the master file access. The two master files are read from beginning to end and the updated versions are written out.

The processing is done on a fund by fund basis.

(a) The fund record is read. The funds’ trailers are spread out in core for ease of reference. Any transactions relating to the funds are processed. The fund record is retained in core during the course of (b) as it may need to be referred to during member processing.

(b) The records for every member of the fund are read, and re-written after any transactions required for the member have been processed.

(c) The fund record is re-written after all the members have been dealt with.

Separate programming modules to deal with the transactions are superimposed by the use of the overlay system. A section of core is set aside for transaction programming and phases containing the appropriate modules are overlaid into this area as required.

One phase may contain modules for several transactions. The modules are arranged in phases according to their most common usage and the amount of core they require. The idea is to minimize the number of different overlay phases to be called in as this can be time consuming.

In particular, keeping different types of fund on the same files leads to a multiplicity of transactions and overlay phases to deal with them. This is not a problem in itself because it leads only to overlay phases being called in on change of fund. Timing problems arise when combinations of transactions requested in the one run require several overlay phases being called in for each member.

At present, as a transaction code is detected, reference is made to each of 3 lists kept in core to determine the action required for the transaction.

1. A list of transaction codes and the overlay phases required for them.
2. A list of fund transactions which require some processing for each member of the fund (for instance, a request for a complete list of members).
3. A list of fund transactions which must be done after the member processing for the fund rather than before it.

Keeping those lists in core involves recompiling the file maintenance program every time a new transaction is added, but for the number of transactions likely to be in use, this would be less trouble than keeping the lists on a separate file.

One consequence of holding up transactions until all errors are resolved is the possibility that no transaction processing at all will be required in a particular file maintenance run. Rewriting the master files in this case would duplicate current versions of the file and some back up would be lost if the new file is to be written over the oldest version of the file. To avoid this, if no transaction processing is required in file maintenance, the files are not re-written, and messages are produced indicating that the latest versions of the files are on the same tapes as before the run.

2.3 Output Analysis

A general printing program which produces all the required lists from the output messages generated in file maintenance:

In an integrated system output messages are produced out of the order in which they are required. This is because the processing is done fund by fund and member by member rather than by one complete transaction at a time.

The output messages must be sorted. There could be...
a separate program to deal with each type of message. This approach may be justified if the system becomes more complex, but provided all output is in the form of printed reports, then a single general program incorporating a sort is ideal.

Each output message describes one line to be printed and consists of 3 parts.
(a) A Sort Key — including stationery type, report type, policy number, line type (heading, sub-heading detail, etc.), any other sorting information required.
(b) The line as it is to be printed — 132 characters.
(c) Other Information — including carriage control indication, amounts to be added to page totals if these are used, indications of footnotes required.

The program is quite general and has been used in other systems. It incorporates facilities for specifying number of copies of a report, accumulation of page totals, footnotes to be included depending on the detail content of the page, requests for the operator to load different types of stationery.

2.4 Apart from any output resulting from the transactions there are 3 lists procured in every run.

Validation Error List
List of transactions processed
List of transactions held on the transaction holding file.

COBOL is used throughout as there is nothing scientific about the programming required. This enables even the file maintenance program to be easily understood.

The largest program, file maintenance, uses 80K and has a 100K overlay area. With some 350 funds on the master files, a complete run with an average number of transactions takes approximately 80 minutes on an IBM 370/158.

Four generations of the master files are kept, the oldest one being overwritten in each run. It has been found that at least one run a day must be made available, if requested because of the reruns needed to clear all the errors for a fund.

3. TRANSACTIONS

Each possible request for processing on the files is given a code number and referred to as a transaction. Member transactions operate on the member file. Fund transactions operate on the fund file although they may also require processing for each member.

It is possible to separate requests into three types of transactions:
1. Alterations — Setting up of new information and replacing existing information by data read straight from the input.
2. Calculations — This is the area which provides the relief of manual work. Any calculation can be cast in the form of a transaction provided the rules are laid down. It is always possible to fall back on alteration for the odd case where the rules break down.
3. Presentation of information — So far this has always been in the form of printed lists.

It is most important to resist the temptation to include all aspects of programming for one application in the one transaction. In particular 2 and 3 should be strictly separated to enable interchangeability between different calculating and printing requirements.

Transactions should be available in units as small as practicable. The variety that this provides is important when dealing with managed funds. If certain combinations of transactions are used frequently it is simple to form a composite transaction which should be available in addition to each individual unit.

This composite transaction uses a different overlay phase, but its overlay phase would consist of a string of the same programming modules used for the unit transactions. In this way any alteration in one of the modules will be reflected almost automatically in the composite transaction.

As an illustration to this point the following is a list of transactions nearly always required when a new member is added to the fund.

alteration transactions setting up name, date of birth, salary and category of membership,
calculate contribution,
calculate death cover,
calculate insurance charge on death cover.

These transactions are available, individually, and in addition there is a composite “new member” transaction, enabling the user to request all these functions by specifying just one code.

There is room in the fund and member files for one transaction and a date to be held internally. When this date is reached, the transaction is processed without any need for card input. This facility can be used for diary notes and for automatic production of a member list to start the ball rolling for an annual review.

Fund transactions and member transactions have been referred to. There are also blanket transactions which operate on every fund. The transactions codes are chosen so that, for the most common cases, transactions to be done first have the lowest numbers. To override this order in the more unusual cases, a priority number may be included. This results in new fund and new member transactions having the lowest numbers, followed by other alteration transactions. Type 3 transactions have the highest numbers.

The transactions to change member name and date of birth deserve special mention because of the problem mentioned in 1.1.

Each of these transactions is transformed by “Validation” into 2 transactions; one with the old key to enable the existing record to be deleted, another with the new key and containing a complete copy of the existing record so that it may be set up in its new position in the file. The sort at the end of input edit places these two transactions in the correct order for processing. Transaction input is in the form of cards which are punched from specially designed input documents. The card formats are as standard as possible, but about half the card contains information which varies by transaction.

As an illustration the following are a few examples of transactions available for money purchase plans, one of the common types of managed fund.

Member Transactions:

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M150</td>
<td>Set up new member</td>
</tr>
<tr>
<td>M151</td>
<td>Change member name</td>
</tr>
<tr>
<td>M152</td>
<td>Change member date of birth</td>
</tr>
<tr>
<td>M350</td>
<td>Delete member</td>
</tr>
<tr>
<td>M550</td>
<td>Alter selected item(s) in member standard information</td>
</tr>
</tbody>
</table>

The Australian Computer Journal, Vol. 9, No. 4, November, 1977
Integrated System for Managed Funds

2. Calculations
M620 Calculate a member's contribution
M656 Calculate the credit standing to a member's account in the fund
M658 Calculate the total contributions a member has paid
M661 Calculate a member's death benefit and the amount of this benefit which is insured
M664 Calculate the charge for the above insurance
M680 Determine what medical evidence is required in respect of a member.

F680 Perform M680 for all members of the fund

3. Presentation of Information
M8xx Print any one of a variety of reports for an individual member — the type of report depending on "xx".

F8xx Print any one of a variety of reports showing details for each member of the fund. The format of the report and details to be printed depend on "xx".

4. Composite Transactions
M750 Perform all processing required for a new member, incorporating M150, M620, M661, M664, M680 and an M8xx transaction to provide a printed record of the new member.

F750 Perform, for each member of the fund, the calculations required for the fund's annual review incorporating F620, F656, F658, F661, F664, F680 and a several F8xx transactions.

CONCLUSION:
In considering mechanizing a managed fund system, the exceptions to the processing rules are the problem. There are so many of them. This has lead some offices to keep manual systems for this type of business. These exceptions can be catered for provided the file access programming is separated from the application programming, (i.e. the program modules required for individual transactions). To take this separation to its logical conclusion there seems to be a good argument for having much of the application programming, particularly the calculations, done by the clerks who would work the calculations under a manual system. For simple applications it may be simpler for such a clerk to code a few COBOL statements than to explain the calculation requirements to a programmer. Everything requiring specialised computer knowledge has been dealt with in the system design and file access programming. Program modules required for individual calculations would then fit into the overlay area like program cards into a programmable calculating machine.

An integrated system offers continual development with a minimum of effort and a very much simpler operation than other types of system. It makes mechanization attractive under present conditions where there is a variation in every little rule from one fund to another. If offices aim for some form of standardisation of managed funds in the future, the advantages of the system will be even greater.
The Implementation of a Data base Management System

By H. G. Mackenzie* and J. L. Smith*

This paper briefly traces the history of FORDATA, an implementation on CDC Cyber computers of the Codasyl Database Task Group’s 1971 recommendations. It describes the characteristics and implementation of the various components of the system, and gives reasons for particular design decisions, and suggests possible system extensions.

KEYWORDS AND PHRASES: Computing Reviews Categories: 3.70, 3.73, 4.22, 4.33

1. INTRODUCTION

The Codasyl Database Task Group, known as DBTG, published reports in 1969 and 1971 (Codasyl, 1969, 1971), which specified a set of proposals for Database Management Systems. These proposals have produced a great deal of discussion on the most desirable architecture for Database systems, and have also been implemented by a number of manufacturers and other groups. We shall assume a basic familiarity with the Codasyl DBTG architecture; (Taylor and Frank, 1976) provides a good introduction.

The FORDATA system began with an implementation on a CDC 3600 computer by one of the present authors (J. L. Smith). It was originally based on the Codasyl DBTG’s 1969 report, and was described at the fifth Australian Computer Conference (Smith, Chmura and Johnson, 1972). The present paper documents the changes that have taken place in the system over the last five years, during which the system has been primarily available on the Cyber76 computer system.

The present FORDATA system is based on the Codasyl DBTG 1971 Report (Codasyl, 1971). It consists of two Data Description Languages, two Data Manipulation Languages, and a Utility Package. There is one Data Description language, or DDL for schema, and one for subschema definition. There is one Data Manipulation Language, known as DML which provides the data manipulation commands defined in (Codasyl, 1971). This DML uses Fortran as its host language. A preprocessor converts Fortran augmented with DML commands to an equivalent Fortran program with calls to library routines in place of DML commands. It also inserts common blocks containing the user working area and the system communication locations, and other necessary Fortran declarations into each routine. As the listing produced by the DML translator includes line numbers which refer to the generated Fortran code, it is unusual in practice for listings of the generated code to be requested.

The second Data Manipulation language, known as QML, is designed for high level query only. No update commands are provided. It is implemented as a preprocessor which generates Fortran augmented with DML commands, and this must then be processed with the DML translator. Thus a query expressed in QML must pass through three translation stages, QML, DML and Fortran before it can be executed to produce output. This does not produce any practical complications as far as use is concerned, and the implementation was made much easier by using this cascaded preprocessor approach.

The system falls short of being a complete implementation in a number of areas.

(1) It does not provide a COBOL host language interface. This is not as serious as it might appear, as an overwhelming proportion of the programming in CSIRO is done in Fortran.

(2) There is no comprehensive multiaccess control system (discussed later). Thus the KEEP and FREE verbs are not implemented.

(3) The MANDATORY/OPTIONAL property of set members is not implemented, a legacy of (Codasyl, 1969). The effective default is OPTIONAL.

(4) The PRIVACY and CHECK clauses in the DDL are not available. Procedural data items, specified by a RESULT clause, may be used instead in many cases.

(5) There is no Data Dictionary/Directory system. Although this may not be a component part of many present day Database Management Systems, it is often used in conjunction with them.

(6) The ORDER verb is not implemented.

(7) The available interactive system does not allow the full potential of database usage to be realised, although we expect this to be improved shortly.

2. ADDITIONAL FEATURES

In addition to the QML, several other improvements to the specifications have been implemented.

An extra DML command, (CHANGE), to augment the location mode specification, and exercise better control over record placement at run time has been implemented. This command also allows the user to inform FORDATA if he is presenting the records to be stored in a sorted set in sortkey order, allowing the time spent in searching the set or the index to be saved.

The currency suppression clauses in the STORE and FIND commands have been extended to allow the user to specify those sets whose currency updates are not to be suppressed. This has also proved very useful in practice.

An extension to format 6 of the FIND statement, to...
allow sets to be searched for a record containing keys equal to or greater than the key values supplied in the user working area, instead of matching on equality only, has been implemented. This enables efficient searches for records with data items within a range of values to be programmed in DML, without the programmer having to search through all members of the set. Any indexes defined on the set will be exploited.

An additional set mode called INDEX mode in which the major storage structure is a B-tree index (see Section 3.3). If the set is owned by SYSTEM there is no other storage structure, otherwise each member must be LINKED TO OWNER. This is the most efficient method of providing secondary indexing to a collection of records of one type which are not at the root level of a hierarchy.

A COBOL like definition for variable occurrence data items, with an upper bound being declared for the number of occurrences. The number of occurrences in any instance is controlled by a separate integer data item.

An extension of the MOVE verb to support the implementation of high level query languages. A list of database keys (a pointer-array) corresponding to a key value for a particular SEARCH or SORT key can be retrieved in a user working area array. Alternatively a count of the number of duplicates can be returned. The whole pointer-array structure of the index or pointer-array set can be traversed in key sort order using this command.

3. IMPLEMENTATION

There are a number of crucial design decisions which have to be made when implementing a Database Management System based on the CODASYL architecture. These decisions are interdependent, and determine the way in which other facilities may be added to the system. Some of the most important facilities involve the following:

1. I/O.
2. The interfaces between Database Management System, application programs and operating system.
3. Storage structure, and the access methods to be supported.
4. The representation of the object schema or subschema.
5. The binding time of the data structure to storage structure.

The considerations and tradeoffs involved in these decisions will be discussed, together with the alternatives, and the course of action adopted in implementing FORDATA.

3.1 I/O

The I/O system was implemented to take advantage of the architecture of the CYBER 76, and the features of that machine which influenced the design of the I/O system will be described. The CYBER 76 provides a hierarchy of two primary memories, one of 64K 60 bit words, called Small Core Memory, or SCM, and one of 512 thousand 60 bit words, called Large Core Memory, or LCM. SCM is used for the storage of executable instructions and searches for which is to be accessed in a random way, whilst LCM is used primarily for large block transfers to and from SCM. It is used by the operating system as a program swapping device. Single words of data can also be addressed, although this capability is not used in FORDATA. In its fast block transfer capability, LCM is similar to the Extended Core Storage, or ECS, available with the lower models of the CYBER range.

The database is divided into a number of areas, each of which is a separate random access file, resident on disc. Space on the discs on the CYBER 76 is allocated in multiples of a number of 512 word sectors. Each database area is divided into a number of 'blocks', which must be a multiple of 512 words, matching the disc space allocation. Each block contains a number of 'pages'. System buffers in SCM and LCM are allocated to hold blocks and pages. I/O operations are done between the area files and the LCM block buffers.

There are relatively large, relatively slow, transfers of blocks to and from area files and LCM buffers, and smaller, much faster transfers between LCM block buffers and SCM page buffers. By having a two level system consisting essentially of pages of two different sizes, it was hoped to effectively use the transfer characteristics of the mass storage devices and the primary memories, and to reduce the amount of SCM necessary.

Due to several requests from overseas installations, FORDATA was modified to run on lower CYBER machines at the beginning of 1976. This necessitated altering the I/O significantly. At the same time a version was developed which used a single level paging system, for installations where Extended Core Storage was not available.

3.2 Application Program/DBMS/Operating System Interfaces

The application program makes requests to the database management system at run time. These requests are obeyed by routines which must access a translated version of the schema or subschema, the database and the user area and the system locations of the application program.

The database management routines may either (1) reside in a supervisory program which communicates with a number of application programs accessing the database, or (2) may be part of the operating system, or (3) may be loaded with each running application program. The first alternative allows control over concurrent update of database areas at either the record or the page level. The supervisory program acts as a monitor for the database accessing programs. If, in addition, the database were reentrant, they could service multiple databases as well as multiple programs. The second alternative is similar to the first, but is less flexible, as in effect the supervisory program is absorbed by the operating system.

FORDATA is implemented using the third alternative. At the time that FORDATA was transferred to the CYBER 76 there was no efficient means of passing messages between running programs, although this has now been implemented (Ryan 1976) and used in an experimental way.

Thus concurrent update in FORDATA is controlled in a gross way, at the area level, by allowing the operating system to prevent simultaneous access to a file which is being updated. This is less important in an interactive environment, and less important in the CSIRO context of databases with only a few users than in the context of an organisation-wide database with many users.

Each application program must have access to a 'user working area' which contains space for each data item defined in the subschema being used. It must also be able to access a set of system locations which are used by the
3.3 Storage Structure and Access Methods

Fixed length records are allocated within pages. The size of any record must not exceed the page size. Each record has a thirty bit header, consisting of the record's size, its type and the number of the record within the page, called its line number. Each record is identified by its database key, a thirty bit quantity allocated when the record is created. It consists of the area number, the page number within the area, and the line number. The database key constitutes a virtual address, which must be mapped into a real address in the system buffers in SCM before the record can be assessed. This operation, together with any associated I/O operations, is performed extremely frequently and hence must be as fast as possible. The routines which implement this mapping are written in assembly code, and are the only parts of FORDATA not written in Fortran. To speed up the address mapping, the offset of each record within a page is kept in an array at the end of the page.

Each page has a three word page header, comprising housekeeping information such as the page type, the amount of space left in the page, a pointer to the next available word, an identification for the last run unit to update that page, etc.

When space is to be allocated for a record in an area, it would be possible to read pages from the area to the SCM system buffers and to compare the number of words available in the page with the number of words required until a page in which the space request could be satisfied was found. This would, however, be very inefficient, as many pages might be read in, only to be rejected. To avoid this, pages called map pages are allocated at regular intervals throughout an area. Each map page is associated with a large number of data pages, and contains the amounts of storage available in each of the associated data pages.

Thus to allocate space it is necessary only to scan the map page entries for a page with enough space to satisfy the request. In addition to making the search for a page with enough space to satisfy an allocation request more efficient, map page entries indicate whether a page has ever been referenced previously. If not, no attempt is made to read the page's block to LCM. Instead, a slot in LCM is created for the block and initialised with empty pages, one of which is then copied to SCM. Thus an area need only have the map pages initialised before use.

Each record contains a thirty bit header containing the record size, type, and line number. It also contains space for all the data items declared in the schema. Records are grouped into sets either by embedded pointers (mode CHAIN), or by arrays of pointers kept in special auxiliary records (mode POINTER-ARRAY). All pointers between records are database keys.

The access methods are determined by the location mode property defined for each record type in the schema, and by the indexing properties defined on each set type. Location modes DIRECT, CALC and VIA SET as defined in (Codasyl, 1971) are supported.

The hashing algorithm provided by FORDATA starts with a quantity containing alternate one and zero bits, and does a bitwise exclusive or of all the CALC keys into this quantity. This is folded, and the result taken modulo the number of pages in the area. There is provision for the user to specify his own algorithm for any CALC record type. This would be done if the FORDATA supplied algorithm gave a highly skewed distribution for those keys actually present in the user's data.

Records with location mode VIA SET are placed in areas as close as possible to the point within the set where they are to be logically inserted. This facilitates sequential searching of a particular set occurrence. This sequential searching may be improved by declaring that an index is to be maintained for a data item or combination of items in the member records of the set. The indexing is specified in the schema either in conjunction with the declarations for sort keys, or by declaring that data items are to be search keys. The indexing technique currently implemented in FORDATA is a variant of the B-tree technique described in (Bayer and McCreight, 1972), and in (Knuth, 1973). This variant, in which records, or pages containing records, constitute the leaves of the index tree, and non-leaf nodes contain only pointers and keys, has been analysed in (Wedekind, 1974). An important characteristic of this indexing technique is that the tree is guaranteed to be height balanced, that is any path from the root of the index to a leaf record is the same length as any other path. This remains true independently of the order in which the records are stored.

There are two types of indexing supported. For ordinary indexing, specified with sorted sets or with search keys, the records may not occur in the area in the order implied by the index, especially if they were not presented in that order. The second type of indexing, called page indexing, is specified in conjunction with sorted sets whose owner is SYSTEM, and maintains the records in the same area order as the sort order of the set. This occurs even if records are presented out of order, although it is more efficient to present them in sort key order. The area ordering is maintained by moving records if necessary, thus changing the record's database key, and violating the rule that database keys are constant over the life of the record. A sequential scan of the set can be done in a minimum number of disc accesses. Sets which are page indexed must have only one member record type, and that member may not be a member of any other sets.

The index in a page indexed set is sparse, in that not all key values are present in the index. The lowest level of the index points to a page which contains member records,
rather than to an individual record. This page is sequentially scanned for a match between keys in the record and keys in the user working area when a record retrieval based on the key values is done. Thus if there are n records in a page, the index size will be reduced by a factor of n. A typical range of values for n would be 20 to 50. This type of indexing is equivalent to IBM's VSAM access method, (Wagner, 1973), and to Control Data's Indexed Sequential, (Control Data 1973).

3.4 Binding Storage Structure to Data Structure

The stage at which the data structure, represented by what the user defines in his schema and DML commands, is bound to the storage structure, that is the pointers, records, packed data items, indexes, calc chains, etc. on physical devices, is a factor affecting both performance and ease of implementation. It would be possible for the DML translator to generate code particularised to each database, which would allocate and delete storage, pack and unpack data items and pointers from records, follow chains and manipulate indexes, and perform other necessary operations. In this compilation approach, record, set and area names would not explicitly occur at the user's object code level. The time taken in interpreting the operation at run time would be minimised. However, this early binding of data structure to storage structure is difficult to implement and change. It also results in some loss of flexibility. In that the user could not write procedures containing DML commands in which record, set and area names are supplied as parameters.

In FORDATA, the DML translator uses the schema or subschema only to check the legality of the CML commands, and generates calls to subroutines which interpret the commands at run time in conjunction with the schema/subschema. This late binding approach offers greater versatility to the implementor. The interpretation time has not proved to be excessive, although it is possible that further extensions to FORDATA would involve generating code at DML translation time in cases where parameter values were explicitly known.

3.5 Representation of Object Schema and Subschema

The way in which the schema is represented at run time must facilitate efficient execution of DML commands. The object schema is represented as a collection of records stored in area zero. It is stored in the same way as the rest of the database, and a number of routines used to manipulate the database are also used to manipulate the

![Figure 1](image1.png)

![Figure 2](image2.png)

![Figure 3](image3.png)
Each data item in the record has its name, type, size, user working area location and information for packing it into the stored record encoded within an ITEMINFO record. If the item has a source item in the owner record of a set for which the record is a member, then the SOURCEINFO record, containing the name of the source item in the owner record, and a pointer to the owner's RECDESC record, is present.

If the item is a procedure data item then a RESULTINFO record, containing the information specified in the RESULT clause of the DDL, will be present. The procedures which generate these data items are written in Fortran and DML, and may themselves cause other procedure data items to be generated. The RESULTINFO record is available to the procedure mainly for cases where the procedure is parameterised to be used in generating more than one data item.

If the record has location mode CALC, there will be a CALCINFO record for each of the CALC items defined in the DDL. Each CALCINFO contains a pointer to the ITEMINFO record for that item.

Sets of records are present for each set owned by the RECDESC in question, and for each set for which the RECDESC is a member. These OWNEDSET and MEMBERSER records contain the information necessary to manipulate the sets. Additional records are associated with the OWNEDSET and MEMBERSER records (fig 3), and provide information on indexing, set occurrence selection, sort keys, search keys and alias data items.

Many records in this structure may be accessed during the interpretation of a single DML command. To make this accessing more efficient, many of the object schema records contain pointers or arrays of pointers to RECDESC or ITEMINFO records.

Representation of the object schema in this way has been very convenient as far as implementation is concerned. One disadvantage is that, as the schema is itself a collection of records in an area, it competes for space with the user's database pages in the system buffers and parts of it may be swapped to backing store. It also incurs overhead in that some pointers in the schema are database keys which have to pass repeatedly through the storage mapping process.

3.6 Translators

The input languages for the schema and subschema DDL, and the DML, QML and Utility packages are all defined using regular grammars. The languages were each specified as a set of nodes, (representing states), and arcs, (representing transitions between states). Each arc is labelled with an input symbol and an action to be performed, (for example consistency checking or code generation). Each translator has a starting state. If one of the arcs leading from that node was labelled with the current input symbol, then the action specified on that arc would be executed and the arc traversed to obtain a new current state. A package, SSTRAN, which enabled recognisers of this sort to be specified extremely easily is described in the appendix.

After a certain amount of user experience had accumulated, it became evident that the DML as defined by the DBTG was at too low a level for quick production of programs. A higher level query language, called QML, was implemented to overcome this. The QML gets away from the very procedural, single record at a time logic of the DML. The user specifies a path through the schema diagram by naming records, data items or sets, and can specify constraints to be satisfied at any point in the path. Canned loops containing DML statements are generated to retrieve the data items specified explicitly or implicitly along the path. Each instance of the path is retrieved, as if the data items were stored in a tabular form. The path may be in the form of a tree. The data items are either returned to the user working area, or are printed, or are output to a user specified file for further processing. The QML package includes verbs to sort a generated file on specified items, to merge files, and to generate reports. Although it is still necessary to know the schema structure (in terms of its Bachman diagram), the production time of retrieval programs is greatly reduced. Of course the user may also write a program containing a mixture of QML, DML, and Fortran.

The Subschema translator allows the user to select a subset of the areas, records, data items and sets defined in the schema. Areas may be specified as being useable for retrieval only, and in this case subsequent attempts to open them with update permission will be rejected. If the subschema specification does not include structural items that are essential for update, then the area containing those items will not be allowed to be opened for update under that subschema. For instance, for update to be possible, if a record is included in the subschema, then all sets for which that record is an automatic member must also be included.

3.7 Utilities

A utility package which allows the user to perform such functions as selective printing, dumping, auditing, increasing the length of an area, and recovery from database corruption, is available.

At INVOKE time, it is possible to specify that journal files of pages before or after modification be maintained. Each page on a journal file contains a number identifying the run in which it was written, and a journal file containing a history of runs may be kept. The Roll Back utility takes a journal file of pages before modification and restores a possibly corrupted database to a state at which it was known to be correct. This would be used in the event of a program or system abort, or a program logical error. The Roll Forward utility uses a dump of the whole database or area and, using a journal file of pages after modification, moves the state of the database forward to the last state at which it was known to be correct. As well as being used in the previously mentioned situations, this recovery method would be used where total loss of disc files had occurred.

4. CONCLUSION

In Section 1 a number of significant features of the CODASYL specifications which were not implemented were listed. The provision of a FORTRAN host system instead of the specified COBOL host system was a decision based on predicted applications. FORTRAN is the most frequently used programming language amongst CSIRONET users (a situation reinforced by the Cyber installation), and the majority of applications of FORDATA have dealt with scientific databases. In these cases a requirement for privacy control beyond that provided by the SCOPE operating systems did not exist. The system could easily be extended to provide a COBOL host interface, but some difficulties would undoubtedly arise at the subschema level with the present choice of
DDL.

Except for demonstrations on a lower CYBER system our experience and total development effort has been with the SCOPE 2 operating system on the Cyber 76. The characteristics of SCOPE 2 were a major influence on the design of FORDATA and forced a number of undesired limitations. In particular in order to control a general multi-access situation the FORDATA system would have to operate in a supervisory mode as described in section 3.2. SCOPE 2 provides no efficient means for passing messages between executing processes (although this has now been implemented by Ryan (1976) and used for other local extensions to the operating system. Thus the KEEP and FREE verbs which form part of the CODASYL multi-access control were not considered. Interactive database access has been limited for similar reasons.

Other CODASYL specified features were excluded as a compromise based on likely use and a desire to limit the workload in implementation and maintenance. For example the CHECK and ON clauses have an important role in database integrity, but it was thought that these controls were much more appropriate in situations where databases would be maintained by numerous application programmers. This has not been the case in CSIRO scientific databases. A number of facilities specified in the ORDER verb were more conveniently provided by a SORT command in the QML.

For similar reasons no effort was expended in producing translators which generated efficient code by a fairly complete binding at translation time. In a Cyber 76 environment the CPU has a bandwidth to main memory somewhat in excess of 200 million bits/sec. In contrast the common database storage medium of demountable disc packs has an average access time of 30 m sec. and a transfer rate of 6.8 million bits/sec. In addition the SCOPE 2 scheduler does not accommodate I/O bound jobs well. Therefore from a global optimisation viewpoint there was much more to be gained by expending our effort in providing a range of access methods and secondary indexing, and multi-level paging, in order to reduce disc storage I/O. The exclusive use of execution time binding has proved advantageous in allowing continued evolution of the system without effecting existing databases and application programs. The most serious disruption has been to call for a retranslation of source schemas to coincide with a particular system update.

The performance of FORDATA is discussed in detail in (Smith 1976). We feel that it is fairly good at present, although there are areas of the implementation which could be improved significantly as indicated above. The additional features described in section 2 provided some important enhancement, for our purposes. We believe that those changes reflected at the language level would be valuable provisions in any similar system. In the light of the more recent ANSI-SPARC architecture some of these features would be properly contained at the internal schema level.

Several companion papers in this issue describe the experiences of some users with the system. The paper by Cook and Mayo (Cook and Mayo, 1977) describes a relatively small but quite complex database relating to a land use research project. The paper by Shortridge (Shortridge, 1977) describes a large library retrospective search database, which at present occupies 400 Mc. and which when fully developed will occupy well over 1000 Mc. The paper by Mackenzie and Kelly (Mackenzie and Kelly, 1977) describes a prototype abstract retrieval system, written to investigate ways of efficiently performing boolean operations on a DBTG structured database.

Appendix: SSTRAN, A translator writing aid for Regular Grammars

The languages in the CODASYL DBTG proposals have each been defined using a regular grammar. A syntax analysis for such languages can be based on symbol state tables (Day, 1970). During syntax analysis the current state is determined by some number of ordered symbols occurring immediately prior in the input program stream. The legality of the next input symbol is completely determined by the current state. In processing this symbol the analyser moves to a new current state. Such a finite state machine is conveniently represented by a transition network (Conway, 1963), the nodes representing the possible states and the labels on the directed arcs representing the symbol which causes the corresponding state transition. By incorporating an appropriate action with each state transition of the analyser, a translated form of the input program can be generated as an output stream.

SSTRAN is a compiler-compiler for generating such translators, implemented for the Cyber range of computers. It requires two input files, one being a table of reserved symbols, and the other specifying the required finite state analyser in a simple tabular format. The latter has a label field and a branching condition so that identical pieces of a state transition network need only be specified once. When using the branching facility a return label can also be specified, thus providing a subroutine or recursive facility. The user of SSTRAN must also provide a routine to read the input program stream, all error and other message generation routines, and the action routines.

SSTRAN was written by D.M. Ryan specifically for use in the FORDATA project. It has been in a number of other language implementation projects.

References

CODASYL (1969) "Database Task Group Report, August 1969". Available from ACM.
CODASYL (1971) "Database Task Group Report, April 1971". Available from ACM.
CONTROL DATA (1973) "Record Manager Ref. Man." Available from ACM.
RYAN, D.M. (1976) "Communication Between Jobs on the Cyber 76". DAD p. 714. (Division of Computing Research Internal Publication.)
Memory technology has undoubtedly had a profound effect on the rate and direction of the development of computer systems. The economic manufacture of large memories (and fast memories) has only resulted from vigorous research and development of a whole variety of mechanisms which have been recognized as potentially useful for making memory devices. In this book the authors present in some detail, the physical basis of currently important memory types, and catalogue quite a few others which have either flourished briefly and died, or which are at present undergoing development with the hope of eclipsing or complementing today's technologies. It is as well to keep in mind that this book is concerned not with the design and performance of memory systems but only with the physical behaviour of materials and the relationship between physical properties and the parameters of the basic memory elements. For each class of device the basic principles of operation are first described and generally a brief outline of manufacturing methods is given. This is followed by an analysis of the effect of physical properties on device parameters and any important trade-offs.

The first class of devices described are delay line memories. The types described are bulk or surface acoustic wave devices. The second and by far the largest section (over half the book), is devoted to matrix memories. Of the magnetic types, ferrite core memories are covered in detail with shorter sections on thin film and bubble technologies. The next major part of this chapter describes semiconductor memories including bipolar and PMOS processes for RAM, ROM and shift register organizations. Inevitably this part of the book has dated rapidly - at the time of writing 1K bit dynamic RAM was the state-of-the-art. The NMOS process gets a mention as do FAMOS, MNOS and amorphous (Ovonic) RAM, ROM and shift register organizations. In Part 2 of the book is dominated by these methods though it also includes a chapter on pole assignment using state and output feedback. Part 1 introduces the basic concepts of controllability, observability and stability rather briefly, and in one instance, laboriously, in § 3.1.1.2 where a one page proof is given for the controllability of a diagonal cannonical form. The author does not make any comments on the relative merits of the numerical methods of evaluation of $\exp(At)$ except to say that the exponential series method is not recommended. It is well known that one can obtain a good approximation to the matrix exponential by using a few terms in the series after ensuring that $\| A \|_1$ is suitably scaled. In Part I there is no reference to literature in the first three chapters, which is in keeping with the author's aim to “thus avoid the necessity of consulting a large number of disseminated papers ...”, but does reduce the value of the book as a text especially since the author does not treat topics like partial observability, partial controllability etc.

In Part 2, the author presents theories for multivariable feedback controller design in five chapters. Only one illustrative example appears (in ch. 8) in this part to illustrate the application of Rosenbrock's inverse Nyquist array technique. It appears to be a poor pedagogical practice to include unsolved problems (e.g. at the end of ch. 9) when no illustrative example is included on a topic which seems to require a considerable degree of judgement and algebraic manipulation at several states of the sequential design procedure. From the text it is not clear if the process can be simplified by use of an interactive computer program.

Part 3 of the book contains material on optimisation found in most control theory textbooks. The author gives the “working rules” for the Pontryagin’s Maximum Principle in ch. 12 and illustrates these with several examples. In dealing with the continuous, linear, quadratic regulator problem with an ‘algebraic’ Riccati equation, the author does not appear to have stated the requirement of controllability or detectability for the infinite duration of control. There seems also to be no attempt to connect the Liapunov function of stability and the Liapunov function with the cost function of the regulator. The properties of tolerance of non-linearity and infinite gain margin of the optimal regulator are not mentioned.

In conclusion, the author presents a concise view of multivariable control theory often unsupported by illustrative examples and without any mention of application. The value of the book as a text is limited by this deficiency and by the lack of adequate references to literature. This lack of references also, unfortunately, limits its value as a reference book. It is, however, a collection of some widely known and some scattered material which may find a useful place on the bookshelf of an initiated control theorist or as a textbook for a student when supplemented by lecture material and illustrative examples.

Reviewed by Dr. K.P. Dabke, Electrical Engineering Department, Monash University.
A Geographic Data Base Using FORDATA

By G.C. Cook* and K.K. Mayo*

The paper describes the design, establishment and use of a geographic data base using the FORDATA data base management package. The data base, established as a research tool in a land use planning study, has been in continuous use for two years.

KEY WORDS AND PHRASES: data base management, CODASYL, schema design, relational records, many-to-many relationships, data structure, geographic data base.

CR CATEGORIES: 3.14, 3.73, 4.33, 4.34

INTRODUCTION

The South Coast Project was a land use study of 6000 km² on the south coast of N.S.W. with the aim of developing and demonstrating a methodology for land use planning at a regional scale (Anon. 1976, Austin and Cocks 1977). The project demanded the establishment of a data base to facilitate the handling of the data gathered about approximately 4000 regions defined within the study area.

It was decided to store the map data defining the boundaries of the regions in a map data base under a special purpose data base system developed within the CSIRO Division of Land Use Research (Cook, in press), and to store the descriptive attributes of the regions in a data base using the FORDATA package of Smith and Mackenzie (Smith and Mackenzie 1974, Mackenzie and Smith 1977), an implementation of the recommendations of the CODASYL Data Base Task Group (DBTG) 1971 Report (Codasyl 1971). The two data bases were logically linked by including, in each data base, data items which were keys to entry to the other. The linked data base system is described by Cook (in press).

The two data bases were mostly used jointly, to display in thematic map form the result of evaluating land use suitability algorithms (or more commonly unsuitability algorithms) for each region. The map data base (not described here) performed region definition, graphic storage and graphic display roles, while the FORDATA data base held data describing the physical, biological and socio-economic attributes of the regions, and was used to evaluate the suitability algorithms. The design, establishment and use of the FORDATA data base is the subject of this paper.

Planning for the South Coast data base was occurring in late 1973, just as the implementors of the FORDATA package were seeking their first users. This application became one of the first users of the package and some experiences reflect this fact. We were constrained by the then current extent and state of implementation, and had no estimates of likely performance. This was, however, a situation into which we went with our eyes open.

We assume below a general familiarity with the CODASYL data structures, data description and data manipulation languages, and we use the terminology and graphic conventions of the DBTG report. Taylor and Frank (1976) provide a convenient and readable source of this background information.

SCHEMA DESIGN

The data base was to be a tool for land use planning research rather than the basis for a defined information system. The form that the land use suitability algorithms would take was not known although intelligent guesses could certainly be made. It was determined therefore that the data base design should aim to reflect the real world relationships between the entities involved, and be efficient for those modes of use which could be predicted. Since the project would involve data collection in many disciplines by a number of people (more than thirty professionals were involved in the project), a number of meetings of those involved were held at which the logical design of the data base and the data items to be included were discussed and eventually agreed upon. On this basis the schema was designed. As far as possible, each record type corresponded to a single data source, so that data management in the establishment phase would be simplified.

Figure 1 shows the complete schema, incorporating 31 record types and 42 set types.* The data base occupies about 75% of an AREA of 800 000 60-bit words. The numbers of occurrences of each record type are indicated in the figure. Aspects of the schema design are discussed below.

Spatial hierarchies and relationships

The data describing the study area was to be organised about a spatial hierarchy of entities. The single study area was to be divided into regions defined by the study team on a map, and each region would be subdivided for descriptive purposes into recurring landform pattern elements or facets (such as crests, slopes, etc.), too numerous to be individually mapped for the entire study area. Fig. 2 shows a small portion of the map defining the regions (the map was stored in the separate map data base) and Fig. 3 shows the distribution of facets in one region.
Figure 1 The schema of the FORDATA data base

A Geographic Data Base Using FORDATA

Study area

Region

Facet

Descriptive hierarchies and other descriptive data

Some region attributes were considered important enough to justify (and simple enough to allow) structural representation using an intermediate level in a study area to region hierarchy. These were the land tenure classification, the planning zone and the land development status.

The facets, being too numerous to be separately described region by region, were each assigned to one of a number of paradigm landform descriptors, chosen after a field sampling program as sufficiently representative of the range of landforms observed. Facets were similarly assigned to descriptors of typical vegetation communities and soil associations. These facets descriptors thus formed an intermediate level in a study area to facet hierarchy structure, using an additional record type clink to relate an including catchment to an included catchment.

- for those regions which overlap two or more map sheets, a relational record slink must be used to represent the many-to-many relationship between map sheets and regions
- the adjacency relationship requires a relational adjacency record linked through two set types to owner region records
- manual set memberships are specified where the appropriate set occurrence might not be known at the time a record is stored.

A number of points may be made:

- there is only one occurrence of the study area record. This could, in versions of the FORDATA system since 1974, have been replaced by the SYSTEM as owner of the various singular sets
- the nesting of catchments requires a recursive

†Throughout the South Coast Project, the term "functional unit" was used for the mapped area referred to within this paper as a "region".
Working storage

One part of the structure was designed for the storage of the results of computed ratings of land use option capabilities. It provided option records to represent the various land use options and a relational rating record which could record a rating of a region for a particular land use.

Redundancy

Some redundancy at the data-item level was incorporated in the interests of efficiency. For instance:

• the shire, land tenure class etc. of a region were incorporated in the data record as well as in the shire, tenure etc. records to obviate excessive FINDS and GETS for a single data item
• the geology and relief, data items in the data record, were also held as data items in the landform record (facets with similar landform description must therefore belong to regions with the same geology and relief.)

Spare data items were included in some records to allow for descriptors not initially anticipated: an attempt to provide some freedom in an otherwise fixed structure.

Estimation of area size

FORDATA requires the declaration of AREA size (to define the range for the CALCing algorithm). A dump of the schema provided information on the length of each record (information now provided by the schema translator) and it only remained to estimate, with a margin for error, the number of records of each type likely to be needed. This was not difficult in this application, but could present a problem in others. Only one AREA was defined within the data base.

ESTABLISHMENT

Initial experience with the FORDATA system, involving as it did a substantial element of system testing, was confined to establishment of the data from a single map sheet. These data were re-established a number of times, due to bugs in the FORDATA system, before

Figure 2 A portion of the map defining the regions

allowing the specification of sub-sets of facets, for example, by landform type. It will be noted that:

• the soils comprising the soil association (one soil may occur in many associations) are themselves described and linked to associations in a many-to-many structure using a link records.
• the facet record is a manual member in sets owned by the facet-descriptive records.

Other descriptive data, too complex for or not justifying representation structurally, were related simply to the region record. Climate being poorly defined in the study area, climate data was related to the district record. Some of these descriptive records were logically merely extensions of the owner record, in that one and only one member-record occurrence was associated with each owner record occurrence, for example:

• the data record was separated from the region record only because of a FORDATA limit on the size of the object schema description of a record. As the region record figured in so many sets, the number of data items which could be accommodated in this description was severely restricted. This restriction was relaxed soon after this application was begun
• the urban record was separated from the region record primarily through having data items from a single disciplinary source.

Other descriptive records could occur any number of times, including zero, per region occurrence; for example:

• the site record contained data items from observational data recorded at any survey sites falling within a region
• one landuse record occurrence was needed for each different land use within a region.

In two cases, records of several related types are grouped as members within the one set type. For example:

• the rec1, rec2, rec3 records, the first containing general recreational data items and the others being used for shore and water recreation only when relevant.

Figure 3 One region mapped into nine recurring facets
sufficient confidence was built up to proceed further.

Establishment plan

Establishment of the data base followed a simple though not too rigid plan. Each region was identified by map sheet (arbitrarily assigned where a region overlapped several sheets) and number within the map sheet. Data were compiled map sheet by map sheet in convenient data item groups, using the region identifiers as keys. Establishment then proceeded by:

- setting up the records of higher hierarchical level which defined sets into which region or facet records would be linked
- establishing *region* records map sheet by map sheet
- once the *region* records for a map sheet were established, other data for that map sheet were input as they became available.

Some parts of the data structure, provided for in the schema, were never used (this reflects the research environment in which the data base was set up). It was decided not to represent catchments, as had been planned, and the *resource and supply* (supply-demand) records were also unused. Input of some data items was deferred until needed — this included data for the *climate* and *site* records which have never been established but could be at short notice (the data are prepared and input programs written.) (The site data have in fact been used — but outside the data base.) Similarly, the linking of partial regions (formed where regions are subdivided by the map sheet grid) to map sheets was deferred pending evidence of need, which has not arisen; again, this section of the data base could be established at short notice.

Establishment problems

An early problem emerged during establishment: the time taken to *STORE* a *region* record steadily increased as more records were input, eventually reaching a quite unacceptable level. The *region* record had been declared INDEXed within the set owned by the *study area* record, and the problem arose from the inefficiency of the single level indexing then implemented. The difficulty was overcome, with co-operation from the FORDATA implementers, by declaring a two-level indexing structure explicitly in the schema and making some changes in the data structure behind the back of the system; the alternative of going back to scratch was at that stage not an attractive one. An immediate improvement in *STORE* times by a factor of about 14 was achieved. (Entirely adequate indexing facilities were incorporated in the FORDATA system during 1975).

It became evident during establishment that great care was needed with currency control in the more complex situations. For example, the sequence of operations needed to store *adjacency* records, each linking two *region* record occurrences (Figure 4) was:

```
  FIND region - 1 SUPPRESS ALL BUT SET - A
  FIND region - 2 SUPPRESS ALL BUT SET - B
  STORE adjacency
```

Lack of sufficient care in one instance resulted in some *facet* records being linked to the wrong *vegetation* record. The trouble arose when trying to correct some initial errors by REMOVEing *facet* records from one set and INSERTing them into another. A correct sequence would have been:

```
  FIND facet
  REMOVE facet FROM set
  FIND vegetation
  FIND facet
  INSERT facet INTO set
```

which, by the early FIND and REMOVE, ensures that the subsequent FIND facet does not disturb the currency of the *vegetation* record required to define the correct set for the following INSERT.

A difficulty was also encountered in correcting the set membership of *region* records which were AUTOMATIC members. FORDATA did not implement the MANDATORY/OPTIONAL specifications, following rather the 1969 Data Base Task Group report which implied "mandatory" with AUTOMATIC and "optional" with MANUAL. To DELETE and reSTORE the affected *region* records would have involved deletion also of the many records in sets owned by those *region* records, and was thus not an attractive procedure. To meet this need, FORDATA was modified to allow REMOVE and INSERTion of records from and into sets of which they were automatic members, substituting a special warning status for the error status previously specified; i.e. all memberships are OPTIONAL.

Using the data base

Retrievals from the data base fell into two clear groups: those confined to a specified geographic area, and those concerned with an attribute-defined subset of the data. In the former case, the associated map data base defined the regions within the area of interest and specified data base keys for the corresponding *region* records, allowing direct access to them. In such cases, the retrieval would normally require evaluation of a specified function of the region attributes, to define a label to be used by the map data base system to annotate a plotted map. In other cases, entry to the data base was directly to the *study area* record, and thence usually to a subset of regions or facets by way of one of the many paths available.

Until very recently, the data base had not been used interactively; all access was by batched jobs. It was found to be efficient, then, to hold the data base as a compressed serial file when not in use, and to expand it out to randomly accessible form prior to use, using a FORDATA utility program. This practice not only economised on.

---

*The syntax used is simplified for clarity.

---

Figure 4 Adjacency structure
permanent disc space, but also avoided the recovery problems otherwise posed by program or system crashes while modifying the data base. For these reasons, as well as the fact that updates were done in large batches, the recovery procedures provided by FORDATA have not been used.

No case of data base corruption has been detected, apart from the occasional mishap in a running job which could be solved by reverting to the serial file.

All programming for retrievals from the data base has used the FORDATA data manipulation language, embedded in a host FORTRAN, requiring the employment of competent programmers. The FORDATA query language facility introduced in 1976 has not proved very useful in our environment. This is because our retrieval criteria are often quite complex and so is our schema; in these cases the translator has not generated optimum programs. However, some limited experience in tailoring simple high-level command languages for specific applications or sub-schemas using a recogniser (Mackenzie and Smith 1977) promises to reduce the need for skilled programmers.

Efficiency

In many retrieval situations, some pre-processing can effect considerable economies. Two examples can be given:

- where evaluation of a function of region attributes entails consideration of a sub-function of the attributes of each adjacent region, it is clearly efficient to first evaluate the sub-function for all regions concerned, rather than evaluate it anew each time a region is encountered as an adjacent region.
- a function of the soil description is best pre-evaluated for each of the 130 soil associations, rather than at each reference from a facet.

Additional working areas for storage of such intermediate results within the data base would have been helpful. As it was, such results were usually stored outside the data using an indexed randomly addressable file.

The options/rating structure proved somewhat cumbersome for its purpose, simply because of the number of record accesses needed for its use; the otherwise unused *supdem* record was more frequently used to store an array of land use option indicators.

The suppression of those currency updates unnecessary to the task in hand was found, especially in the case of the *region* record, to significantly increase efficiency. In the absence of explicit currency update suppression, each new FIND of a *region* record would cause currency updates for all the sets of which that record was owner or member. The use of suitable sub-schemas (only recently implemented in FORDATA) can achieve much of this suppression automatically, and has the added advantage that the size of the schema, interpretively scanned at each DML command, is reduced.

Problems

Although some data item validations had been done before input, including some cross-validation between related data item groups, this proved to have been insufficient in some respects. Retrieval results continued to show up mutually inconsistent data items which needed to be resolved and corrected. This arose from the independent observation and recording of data item groups by different people, whose definitions and interpretations were not always consistent. This could only have been avoided by much tighter data management control of the project.

FORDATA error messages have not always been found satisfactory. They are sometimes cryptic, non-specific, or non-existent (in that an operating system diagnostic is all that is received).

From time to time, action was needed to keep the data base consistent with the developing FORDATA system. As a rule, re-translation of the schema was all that was required. A change in the system CALCing algorithm was met by explicitly declaring the old procedure name in a revised schema.

One serious problem, yet to be explained, arose at one stage with CALCed records. The symptoms suggested the use of differing CALCing algorithms at different times. The solution was to MODIFY all the CALCed records, thereby forcing their location using the correct algorithm.

CONCLUSION

Despite the problems, FORDATA has been a highly satisfactory data management package for this application. The system has proved remarkably resilient in allowing a data base, constructed early in the life of the system, to continue successfully through major system changes and upgrades, without the necessity for re-establishment. The FORDATA implementors, Smith and Mackenzie, have been extremely helpful in assisting with problems and responding positively to suggestions and criticisms.

REFERENCES


Development of a Bibliographic Search System Using FORDATA

By J.D. Shortridge*

The design of a family of databases, to be used in a retrospective literature search system currently being implemented by the CSIRO Central Library, is described.

CR CATEGORIES: 3.74, 3.51

1. Background

Since 1971 the CSIRO Information, Library and Editorial Service has been running a computer-based “current awareness” service in various scientific fields (CILES, 1974). In this system tapes containing bibliographic information, typically references to citations of journal articles, etc. are purchased on a regular subscription basis from overseas, and subscribers within Australia submit “profiles”, or computer-readable descriptions of their areas of interest. These profiles are matched against the tapes as they arrive, and cards containing apparently relevant document references are sent out to the subscribers.

A typical profile is shown in fig. 1. An output from this profile is shown in fig. 2. Profiles consist substantially of one or more (normally two or three) groups each of one or more terms, where a term is a string of alphanumeric characters of interest to the subscriber. Terms may be “truncated” — if a term in the profile ends with an asterisk.

*ROICODEI WA (specifies accounting information)
*N Dr. A. Jones
*D MODELLING IN HYDROLOGY
*M 999 (maximum number of hits to be printed)
*P K NOT SNOW*, ICE, THERMAL, POLLUT*, GLACI*, LAKE*, WAOS* (a SWRA classification)
*Z

Figure 1

Various refinements are possible. For instance, a search term may be of the form

CHROM* IGNORE CHROMATIC CHROMATOGRAPH

in which case the search system will match with any term on the source tape starting with CHROM except for either CHROMATIC or any term which starts with CHROMATOGRAPH. A search parameter may also be of the form

NOT CHROM* X-RAY

in which case records from the source tape will not be retrieved if any match is found with any of the terms within this “NOT-parameter”.

Within the profile it is necessary to specify the portions of the source-tape records against which the search terms are to be compared. For instance, on the Selected Water Resources Abstracts tapes (which are the main subject of this paper) searching may be carried out against authors, document titles, subject headings, keywords or section numbers. Subject headings, keywords and section numbers are all assigned by the abstractors who originally prepare the data which is to go on the tapes — one or more section numbers are assigned to each document, just as entries in a library catalogue may be numerically classified. Subject headings and keywords are descriptive words or phrases assigned by the abstractor to the document, the difference between them being that subject headings are taken from a fixed thesaurus while keywords are freely assigned.

The system was first implemented using current-awareness “C.A. Condensates” tapes, which are, in effect, a computer-readable version of the journal Chemical Abstracts. At this time, eight tape services are available for...
searching, covering the fields of chemistry, physics, electronics, computers, biology, agriculture, food science, and water resources. Since the service was first introduced each tape has been processed as it arrived and has then been stored with the intention that ultimately all the old data would be used to build up a system which could be used for "retrospective" searching of the data (which would be prohibitively expensive using the current-awareness system). A "retrospective" system has now been implemented which permits retrospective searching of the 90000 document references in Selected Water Resources Abstracts (SWRA) (a publication produced by the U.S. Department of the Interior) and a similar prototype system has been implemented for about 80000 Chemical Abstracts (CA) records — this latter corresponds to only three months' data and so is not of much practical use, but it is hoped to extend the system to cover, say, the last four year's data, which consists of about 1.5 million document records. The remainder of this paper will describe the structure, creation and searching of the SWRA database, and will then describe the differences that will occur in other similar systems (particularly the CA system), and make a few general comments on the experiences gained while developing the system using FORDATA (Mackenzie & Smith, 1977). It was intended that the implementation of the SWRA system should, as well as providing a practical service, provide experience of user reactions and various methods of operation which will be useful when search systems are implemented for larger databases such as that based on Chemical Abstracts. The system is currently
available for searching on a batch basis. It is hoped that at some stage users may be able to access the system interactively; however, given the large volumes of data involved and the high costs of on-line storage, this is a medium to long term objective. It is assumed that the reader is familiar with the architecture allowed by such systems. (Taylor and Frank 1976) provides a good introduction.

2. Database Design

The database implemented for SWRA consists of 12 areas, which occupy two 100 Mc disc packs. The database structure is very simple, and is outlined in fig. 3. One disc pack is taken up entirely with five areas, each of which contains the abstract texts for two years' data. The basic reason for storing abstract texts separately is that their provision is experimental and may be discontinued in future, depending on user demand. As the searching process is very fast, a major cost component of a single search is simply the cost of mounting the disc pack. Consequently, a search in which the user requests provision of abstracts on the output will cost about twice as much as a search where he does not, and if it appears that users don't want this facility it will not be included in future systems. (Taylor and Frank 1976) provides a good introduction.

Figure 3. Database structure for SWRA database.

![Database structure for SWRA database](image)

Each 'RABST' record has associated a collection of zero or more 'RABSTOV' records, which are used for storing characters 601-800, 801-1000 etc. of the abstract text where these are required.

Five similar areas on the other disc pack hold the citation data — that is, title, author, subject headings, keywords and document numbers for the various entries. In these 'ACITN' records, previously-encountered subject headings are replaced by a numeric code. The records also contain the database-key of the corresponding abstract text record. Once again, lengthy citations are stored using an overflow record ('RCITOV'), where zero or more of these are associated with each 'RCITN' record.

The other two areas, held on the same disc pack as the 'ACITN' areas, contain the indexes which are searched to find the articles corresponding to a given search profile. In the 'AINDXTM' area (the 'AAUTHOR' area is similar) all indexing terms are stored in a sorted page-indexed set (FORDATA's equivalent of an indexed sequential organisation). Indexing terms for SWRA are fixed-length of 30 characters (this length being chosen as it is desirable to work with fixed-length terms and when the schema was defined at length was selected which would accommodate the large majority of cases). Associated with each term is another item which consists simply of the first four characters of the 30-character term, and this item is used as the sort key. This is necessitated by the requirement for "truncated" search terms — if a term like LAKE* was sought it would not be possible to do a FORDATA 'find' if the set of index terms was sorted on the full 30 characters, since the 'find' operations look for an overall item match. This means, in passing, that while non-truncated search terms can be of any length, truncated search terms cannot be shorter than four characters. From the current awareness system's file of search profiles it seems that these restrictions will not in fact inconvenience anyone.

Each 'RINDXTM' record has associated with it one or more 'RCITN' records — these each contain up to 50 database-keys, which correspond to 'RCITN' records indexed by the corresponding index term. The choice of 50 as the "blocking factor" here involved a trade-off between the overhead that would be incurred by having large numbers of records for the more common indexing terms where a small blocking factor was selected, and the overhead that would be incurred by having much wasted space within each record for the less common terms where a large blocking factor was selected. In fact, with the benefit of hindsight, it would probably be better to store a small number (perhaps 5) of database keys within the 'RINDXTM' records, and use the 'RCITN' records only where a term has occurred in, say, more than 5 documents. This practical consideration arises because of the typical frequency distribution of the index terms in this sort of application — for instance, in the case of SWRA there are 170000 separate index terms, of which 110000 occur just once, while in the case of CA there are 44000 separate terms of which 24000 occur just once and 10000 just twice. In the long term it may prove to be uneconomic to keep these 'uncontrolled' keywords as indexing terms. This is another area where we are looking to the SWRA system to provide experience which we can use in future when dealing with larger volumes of data. There are, however, very good reasons for attempting to keep these terms unless it becomes obvious that it is economically impossible to do so. By leaving them out we would be effectively throwing away some of the information, which may be of interest to a user. For example, as many of the singly-occurring terms in the SWRA database are geographical, a user could submit a search request to see if anything had been published away some of the information, which may be of interest to a user. For example, as many of the singly-occurring terms in the SWRA database are geographical, a user could submit a search request to see if anything had been published involving some small Australian waterway. If singly occurring terms were ignored this search might not be possible. More importantly, certain of the tape services (Chemical Abstracts in particular) do not offer any descriptors from a restricted vocabulary, so the elimination of infrequent terms would be impossible to do safely, given...
the highly dynamic nature of scientific vocabularies. A crude automatic process would be likely to discard the first few occurrences of a significant new term, while a great deal of effort would be involved in trying to maintain a normal thesaurus over such an enormous area.

3. Database Operation

It can be seen that the documents associated with any given search term can be found by executing a ‘find’ to get the appropriate ‘RINDXTM’ record (the fact that the index-term records are held in a sorted set makes this process very fast), extracting the corresponding database-keys which can then be used directly to find the corresponding citation records — these in turn contain the database-keys of the records which contain the corresponding abstract text, which can then be extracted if required. This process, and the process of creating and updating the database, will now be described in more detail.

The application is quite atypical in that there is no requirement to delete records from the database at any stage. When all 5 areas (of abstracts and of citations) are full and it is desired to add new data to the database the oldest data must be removed, but this is done simply by purging and re-initialising the entire relevant areas, after which new data can be added to them (so the database in the SWRA case will always contain the most recent 9 or 10 years’ data). The selection of this period is arbitrary, and could easily be changed, but in any case some limit will eventually have to be set on the time that references are kept in the system. Feedback from users regarding the relevance or irrelevance of their retrievals will be useful here. The process of adding new data to the SWRA database is carried out annually and consists of the following stages:

(i) If necessary (every second year) the oldest existing ‘AABST’ and ‘ACIT’ areas are purged.

(ii) The sequential files as originally purchased are read and the abstracts are stored in the relevant area. Another sequential file is produced, which is essentially the same as the input sequential file except that the abstract text has been replaced by the database-key of the corresponding abstract record in the database. This stage and (iii) following could be carried out simultaneously. The only reason that they are performed sequentially is that the installation policy currently in operation does not allow users to have two demountable disc packs mounted simultaneously.

(iii) This sequential file is then read and by a similar process the citation records are stored in the relevant area. At this stage the terms under which the citation is to be indexed are identified, and these terms are written out to a sequential file as individual records in conjunction with the database-key assigned by FORDATA to the relevant citation record. Typically, there are 15 or 20 of these index-term — database-key records, which are known as “postings”, for each citation record.

(iv) All the postings for the current update are then sorted and held on backup tape (where they will be required for the next 9 years).

(v) This sorted file of postings is merged with all the existing similar files from previous years. The result of this is a file which is, in effect, a complete index to the citation records currently held in the database.

(vi) This total file of postings is then used to build the ‘INDXTM’ area — as each new index term is encountered on the file a new ‘RINDXTM’ record is added to the database, and for each ‘RINDXTM’ record all corresponding database-keys are extracted from the postings file and are stored, in blocks of 50, in the database.

As outlined above, the process of searching the database is fairly simple. Given a search profile, the database-keys corresponding to each individual search term can be found and extracted from ‘AINDXTM’ (or from ‘AAUTHOR’, where searches involving authors are being performed). The resulting collections of database-keys can then be subjected to the Boolean operations that are implicit in the profile structure, and the resulting group of database-keys are those of the citations which appear to satisfy the profile. These citations can then be directly retrieved, and as they contain the database-keys of the corresponding abstract text records these abstract texts can be retrieved in their turn if they are required.

4. Conclusion

It should be possible to implement retrospective search systems for all the tape services using variants of this system, though it doesn’t seem possible to develop a common system for database creation (the search system will always be more or less the same). In general terms, this is because the database creation process is quite expensive and the storage requirements are quite large — each different database will require several hundred million characters of disc space. So it is necessary to cut corners whenever possible, and it is not possible to plan to do this in a consistent way. For instance, SWRA subject headings are taken from a fixed thesaurus, so it is possible to store these as numeric codes. That is not the case for CA data. On the other hand, CA citations on the original tape contain a CODEN, which is a standard 5-characters abbreviation for the journal title — e.g. AJCAA might be the abbreviation for the Australian Journal of Chemistry. So for the CA database a separate FORDATA area is maintained, in which the records contain a CODEN and the corresponding full journal title. The records are hash-addressed using the CODEN, and the result is that CODENs can be used to save disc space and can be replaced by full journal titles just before a record is printed.

The FORDATA system was found to be quite satisfactory for building a database of this type, and it allows very fast searching. A single typical search takes between 2 and 15 seconds, depending on the complexity of the query profile. There were, however, a few “false starts” due to lack of experience in using DBMS systems and resulting original choice of expensive methods of operation. Including these false starts the system took about one and a half man years to develop. As an example, it is interesting to note that the method outlined for building up the index area (by merging all the old postings files) originally seemed inefficient, as every posting is effectively stored several times over its 10-year life. However, not only is this process cheaper for retrieval (due to the fact that the physical order of the index matches the logical order) but it is also cheaper for update. In the update program the ‘CHANGE’ command can be used to inform the system that the elements of a FORDATA sorted set are in fact being presented in a sorted order, thus drastically reducing the processing overheads.
It is hoped that in the future this system can be used to develop a complete set of databases to allow retrospective searching of all the data that has been purchased since the current-awareness service commenced.

The editor of this new series of monographs, L.C. Thomas, describes the object of the series as 'the timely dissemination of essential information about topics of current interest in science', in which interdisciplinary aspects are given fullest attention ... to enable those who are not specialists in a particular subject to appreciate the applicability of the subject matter to their own work, and the bibliographies included in each monograph will guide readers in extending their knowledge of the subject to any desired depth. The depth of treatment of course makes them compact definitive books for the specialist as well.

'The two major defects of this monograph are the almost total lack of reference to structured programming, which is closely associated with the origins of Pascal, and would have been a suitable subject for a final chapter, and the fact that the monograph is based on the original Pascal Report, not the widely accepted ICL 1900 implementation. The four appendices are: a BNF description of Pascal; the CDC 6000 implementation of Pascal, the ICL implementation of Pascal; and a set of exercise problems.'

The Australian Computer Journal, Vol. 9, No. 4, November, 1977
A Query/Update Package for Library or Personal Use

By H. Mackenzie * and G. Kelly **

The implementation of a package to provide access to a database of bibliographic material is described. The package is written using Fordata, an implementation of the CODASYL DBTG proposals. A high level batch or interactive language is provided, comprising an update component and a set oriented query component. Implementation techniques of interest to those implementing similar systems are described, and possible extensions are suggested.

KEY WORDS AND PHRASES: Database, CODASYL, Data Retrieval.
CR CATEGORIES: 3.70, 4.22

Introduction

The package described in this paper was implemented to investigate data organisation techniques for evaluating boolean expressions over the entities stored in a formatted database, in this case one containing bibliographic information, and to exercise the FORDATA system during the later stages of its implementation. It was implemented to run either in batch or interactive mode, although the interactive facilities provided by the operating system at the time that the package was written were rather rudimentary. The actual implementation was done by one of the authors (G.K.) while working as a vacation student, and the combined effort required by both authors was about five man months.

The package was implemented using data on about 1000 books from the Division of Computing Research's library. The paper describes the user's view of the Query/Update language provided, and then describes the techniques used in implementing it.

A User's View of the Query/Update Language

The user views the database as a collection of book or article title entities, a collection of author entities, and a collection of subject entities, called keywords. Each title may be associated with many authors and keywords, and each author or keyword may be associated with a number of titles. A many-many relationship of this type also holds between author's and keywords.

The Query/Update language consists of a set of commands designed for manipulating the sets of authors, titles and keywords and the relationships between them.

The querying facilities search the database for collections of entity instances which satisfy conditions specified by the user. The query commands would be useful in searching a personal reference or abstract library, as well as normal library searches.

The updating facilities are used to modify particular entities in the database. They can be used to correct any erroneous entities or to add new entities to the database.

The user of these updating routines would probably be a librarian.

Not all possible update/query commands will be given here, but some examples will indicate the range and limitations of the commands. The conventions used in specifying the command syntax follow that used in (Codasyl 1971) and in other Codasyl documents.

Query Commands

CREATE

CREATE

AUTHOR

TITLE

KEYWORD

SET <name> WITH <boolean expression>

This command forms a collection of keywords, authors or titles and gives the collection the label defined by the user (<name> above). Inclusion of a keyword, author or title in the collection is determined by whether it fulfills the conditions specified in the boolean expression.

Some examples and their meanings are:

(a) CREATE TITLE SET T1 WITH KEYWORD 'COBOL'.

Form a collection of all the titles in the data-base that are on the subject 'COBOL'. Call the collection T1.

(b) CREATE TITLE SET T2 WITH T1 BUT NOT AUTHOR 'C.A. JONES'.

Form a title collection made up of all the books in T1 above but not including any that were written by 'C.A. JONES'.

(c) CREATE AUTHOR SET AAAA WITH KEYWORD 'COBOL' AND KEYWORD 'COMPILERS' OR KEYWORD 'COBOL IMPLEMENTATION'.

Form a set of authors (and call the set 'AAAA') made up of those books in the database that have both the keywords 'COBOL' and 'COMPILERS' associated with them together with those authors that have written on 'COBOL IMPLEMENTATION'.

On successful completion of an instruction the message SET <name5> CREATED WITH <n> ELEMENTS' is output. It is possible to abbreviate commands by omitting fill words. Errors in command syntax result in an error message, the command being ignored and the next command input. Errors encountered...
while processing a valid instruction (e.g. a specified keyword is not in the database), also result in an error message, the current command being abandoned and a new command input.

It is necessary for the user to know that the boolean operators AND(*)AND NOT,BUT NOT(*) have a higher precedence than OR(+). Hence ‘x AND y OR z’ is parsed as ‘(x AND y) OR z’. It is not parsed as ‘x AND (y OR z)’.

To achieve the latter parse, brackets may be used as indicated. The boolean expression may be of arbitrary length and complexity.

PRINT

Print the named set. All entities in the named set are printed. If the 'WITH' clause is used then keywords or authors or titles associated with members of the set are also printed. The record type specified in the 'WITH' clause should be different from that of the set type. If they are the same (e.g. PRINT KEYWORD SET K33 WITH KEYWORD>), the 'WITH' clause is ignored. Using 'EVERYTHING' in the 'WITH' clause simply causes the set to be printed together with all associated entities of the two types.

Example: CREATE AUTHOR SET A1 WITH KEYWORD 'COBOL' AND KEYWORD 'COMPILERS'. PRINT AUTHOR SET A1 WITH KEYWORD.

The PRINT command will write out every author in set A1, and with each author any keyword associated with that author even if the keyword is not 'COBOL' or 'COMPILERS'.

Update Commands

DELETE

DELETE (K)

(NAME 1 [ , (NAME 2) ... ])

(NAME 3 [ , (NAME 4) ... ] ... )

K,A,T stand for KEYWORD, AUTHOR, TITLE RESPECTIVELY.

The command removes named entities of the stated types from the database, together with any relationships connecting the deleted entities records with other entities in the database. Errors occur, for example, if the named entities are not of the specified type, in which case the remainder of the delete instructions (including the entities that caused the error) is ignored, a diagnostic message printed and the next command sought.

MODIFY

MODIFY (K)

(NAME 1) TO (NAME 2)

(NAME 3) TO (NAME 4) ...

The command changes record '<name3>' to <name4>. For example 'MODIFY A T.S. ELLIOT' TO 'T.S. ELIOT', means all titles previously listed as written by T.S. ELLIOT will be attributed to T.S. ELIOT in the database and T.S. ELLIOT will be removed from the database.

S stands for 'shelf number' which is a property of a title entity. An instruction such as 'MODIFY S 076.420' TO '015.391', changes all instances of shelf number 076.420 to 015.391.

O stands for 'shelf number of title' and is used to change the shelf number of one particular title in the database. An example is: 'MODIFY O 'DIGITAL COMPUTER CONSTRUCTION' TO '123.456'.

ADD

The ADD command has many variations but is used basically to add a complete new set of entities (that is, a title with its associated authors and keywords), to the database and link them appropriately.

There are many variations to this command, many of them being semantically equivalent. Several examples are given below.

Examples

(a) ADD A author-name TO T title-name K key1,key2 ...

This adds a complete collection of entities.

(b) ADD A author-name TO T title-name.

This adds an author to a specified title.

(c) ADD A author-name TO T title-name, K keyword-name.

This adds an author to a title and a keyword.

IMPLEMENTATION TECHNIQUES

1. The Structure of the underlying database

It is not proposed to discuss the FORDATA database architecture which is that specified by the CODASYL DBTG; readers who want a detailed discussion should refer to (Taylor and Frank, 1976). The main features of the particular schema used are described below.

The Schema and Data Items used

The entities that we wish to manipulate with this system are titles, authors and keywords. Each of these entities is associated with a record of the same name containing data items pertaining to the title or author or keyword. Observing that each title has a number of authors, one might be tempted to define a set with a title record as owner, and a number of author records as members. However each author may be associated with a number of titles, and thus the one-many mapping between title and author implied by the set definition is insufficient. One can represent a many-many mapping non-redundantly by having two sets, owned by the records to be related, having as member a common relationship record. Thus a title and an author which are to be related would be bridged by an instance of a relator record (see Figure 1). The FORDATA DML provides commands both for sequencing through the members of a set and for jumping from a member record in a set to the owner of the set.

One method provided by FORDATA for retrieving a record which is not a member of any set is to declare the record to be 'calced' on the values of certain data items
within the record. When the record is to be stored, its address is computed by a randomising procedure using the data items specified. When it is to be retrieved, the same procedure is performed using the supplied key values, and the address recomputed.

The application requires access to any keyword, author or title rapidly, and hence choose to make those records 'calced'. The considerations outlined above lead to the following schema.

Titles, authors and keyword records are related using many-many relationships, each implemented via two sets and a relationship record. Keyword records are related to other keyword records in the same way. Two sets KSUB and KSUPER are included. These relate keywords with more specific and less specific keywords. The title, author and keyword records are calced using the data items, title name, author name and keyword name held within the corresponding records. Thus, given a keyword one can find all the corresponding titles and authors by finding the keyword record and sequencing through the relationships. Similarly given a title or an author one can find the other records corresponding to it, and given a keyword one can find all the more inclusive and less inclusive keywords.

**FORDATA** provides for sets to be specified as being sorted in ascending or descending order using data items contained in the member records.

Data items in **FORDATA** are fixed length fields, but in general the entities represented by data items like title name, author name and keyword name have widely variable lengths. For this reason, each of the main records is the owner of a set containing a record for the overflow text in the case of a particularly long author title or keyword name. This is not shown on the schema diagram. There are other items of information associated with a title, e.g. abstract, publisher, publication date, etc. Again because of the variability of this information, it is kept in an overflow record which is a member of a set owned by the title record.

2. **Command Implementation**

There are two statements in the query language. The CREATE statement, which generates a set of titles, authors, or keywords from a boolean expression involving the other two record types. The CREATE statement is translated into a table, and the boolean expression is translated into a binary tree reflecting and precedences of the three operations set union (OR), set intersection (AND) and set difference (AND NOT or BUT NOT). Given this structure it would be possible to evaluate a boolean set expression by successively creating intermediate sets and using the sets created in subsequent stages of the evaluation. The general method for generating, for example, a set of titles, given a keyword would be firstly to find the keyword record for the particular keyword, and to sequence through the relator records in the '.. . describes . . . ' set. For each relator records, finding the owner of the corresponding '.. . is described . . . ' set will give one of the target title records. All the data items, title name, can thus be formed into a set to be used in further set operations.

The schema outlined above would be extremely inefficient and can be improved in several ways.

Firstly, instead of using the data items themselves from the target records, we can use the database keys of the target records which are unique identifiers for those records. Thus we can manipulate small integer values instead of text strings of arbitrary length during the evaluation of the boolean expression. To obtain these database keys without accessing the target records themselves, we specify that the relationship records contain the database keys of the owners of both of the sets for which they are members.

Secondly, the process of forming a difference, union or intersection between what are now essentially two lists of integers is speeded up by having the lists sorted in ascending order. This is done by specifying that the sets implementing the relationships are sorted on the values of the database keys of the owners of the other set comprising the relationship. For example, the WROTE set illustrated in fig. 4 is sorted on the value of TITLDBK, the database key of the owner of the WRB set. Thus iterating through the AUTHORSHIP records in an instance of the WROTE set would give an ascending sequence of TITLE record database keys.

Thirdly, the creation of intermediate sets can be avoided by evaluating the whole boolean expression in a parallel fashion, using the list set generator technique (Shapiro, 1970). The boolean expression is expressed as a tree (binary to allow a simple implementation), and each node has a process associated with it. A process is an invocation of a procedure containing its own local variables whose execution can be suspended while other processes are invoked, and then continued. The process associated with each node generates the set represented by the subtree rooted at that node. Using Shapiro’s terminology, the set union (OR) process is called a union list set generator (ULSG), the set intersection process (AND) is called an intersection list set generator (ILSG) and the set difference process (BUT NOT; or AND NOT) is called a difference list set generator, (DLSG). The process representing a primitive set is called a primary list set generator (PLSG). Each LSG must be initialised, and may then be incremented to get the next element (database key of target record) for the set that it represents. An outline of the procedure to initialise and increment each LSG follows.

**Initialise PLSG**

1. Find the owner record using the calcing properties of the record.
2. Find the first relationship record in the appropriate set, and get the database key of the target record from it.

![Diagram of data relationships](image)
Query/Update

Increment PLSG
1. Find the next relationship record in the appropriate set and get the database key of the target record from it.

Initialise ULSG
1. Initialise the left LSG.
2. Initialise the right LSG.
3. The lesser of the left and right values is the value of the ULSG.

Increment ULSG
1. Increment the lesser of the left and the right LSGs.
2. The lesser of the left and the right values is the value of the ULSG.

Initialise ILSG
1. Initialise the left and the right LSGs.
2. If the left is less than the right, increment the left, if the right is less than the left, increment the right, repeat this until the left equals the right.
3. This value is the value of the ILSG.

Increment ILSG
1. Increment of one of the LSG's, left or right.
2. and 3. Same as step 2 and 3 for initialising the ILSG.

Initialise DLSG
1. Initialise both left and right LSGs.
2. Increment the right LSG until it is greater than the left LSG. The value of the left LSG is now the value of the DLSG.

Increment DLSG
1. Increment the left LSG.
2. Same as step 2 for initialising the DLSG.

The conditions for termination of LSGs have not been covered here, but are relatively simple.

The use of this technique offers a fast way of executing set operations, at the cost of slightly more data organisation. Note that the target records themselves are not accessed during execution of the CREATE command. There are three relationships implemented in this database, and each of these may be exploited in either direction. The facility of having parameterised DML commands, available in Fordata, allowed each of these six operations to be implemented using a single piece of code.

The PRINT command simply sequences through the generated set of database keys. For each one, it accesses the target record, and outputs the data items to the console or print file. The other records associated with a particular target record are also accessed and output if required.

The update commands are implemented by adding or deleting records and relationship records according to the specified commands.

The language is implemented using a top down, non back-up algorithm. As with the translators used in Fordata itself, it is implemented using the SSTRAN package described in (Ryan 1977).

Conclusion

The package described comprises a basic set of facilities for accessing a structured database of bibliographic material. Other facilities, such as the ability to access the data via more keys (e.g. publisher) the ability to exercise greater control over the format of output and to direct output to other files for extra processing, and the ability to access the author by surname only, would doubtless be necessary for particular applications. The inclusion of commands to list more specific or less specific keywords by using the structure already provided in the schema might also be desirable. Enhancements of this nature could be added without difficulty.

Acknowledgements

We would like to acknowledge the contributions of Dr. J.L. Smith, who provided advice at all stages of this project, and of C. Johnson and Jim Mansbridge, who worked on earlier versions of the package.

References

GINV, A Subroutine in ANSI Fortran for Generalized Matrix Inversion

By H. W. Hodaway*

A description is given together with some illustrative test results for a set of subroutines useful for orthogonalization and factorization of any matrix, with applications to linear least squares problems via the generalized Moore-Penrose inverse. The basic inversion algorithm is dealt with on a column by column basis, which may lend itself to non-linear applications, where progressive updating of columns is used to assist the rate of convergence. Precautions have been taken to minimize the introduction of errors due to accumulated round-off errors of the computer, and some illustrative examples have been tabulated to demonstrate the accuracy capabilities of the algorithm. All algorithms are in ANSI compatible FORTRAN.

CR CATEGORIES: 5.14

Introduction

The generalized inverse of Moore-Penrose for a rectangular or a square matrix (Penrose, 1955; Fletcher 1968) is of general application for linear least squares problems, and can be of value also in non-linear problems of this kind, where its use minimizes the size of correction vectors if the Jacobian matrix approaches singularity.

The subroutine GINV, described here, is in ANSI compatible FORTRAN. It determines the effective rank of the matrix A(M,N) and its generalized inverse A+(N,M). The solution vector x which minimizes the Frobenius norm || A.x-b ||F is given by

\[ x = A^+b. \] (1)

However, it is not necessary for a single problem of this kind to evaluate A+ explicitly, and accordingly the algorithm can give the solution of equation (1) directly by setting a parameter IX equal to 1. However, if IX = 0, A+ is computed and if IX = 2 both x and A+ are computed.

The algorithm employed is based upon one given by Rust, Burrus and Schneeburger (1966) with a number of modifications. Essentially a modified form of column by column Gram-Schmidt orthogonalization is employed to produce the factorization A.Z=Q, where Z is upper triangular and QTQ = I. As each column is processed the error of orthogonalization is checked and, if necessary, corrected. This process is usually very rapid, but as many as three subsequent corrections of the original orthogonalization have been allowed for badly conditioned matrices.

The following are changes that have been made in the algorithm. Firstly, all matrices are stored as single column arrays, and a supplementary subroutine MXTRAP is employed to permit the generalized inverse A+(N,M) to be finally overwritten after transposition on the space originally occupied by A(M,N), for the most usual case where M > N. The MXTRAP algorithm has been translated from an ALGOL procedure given by Boothroyd (1967).

Introduction of errors due to accumulated round-off errors of the computer, and some illustrative examples have been tabulated to demonstrate the accuracy capabilities of the algorithm. All algorithms are in ANSI compatible FORTRAN.

Secondly, in the column by column orthogonalizations a function subroutine DOT is used to calculate inner products in double precision with a final result returned in single precision. This minimizes accumulated round-off errors and, together with the orthogonalization adjustments mentioned previously, is responsible for the achievement of good accuracy in applications.

Thirdly, complete organization in the column by column procedure is extended to include column by column generation of an N x N book-keeping matrix U which is finally overwritten by the matrix Z. This differs from the original (Rust et al; 1966) algorithm, in which U is initially set up as a unit matrix. For application to non-linear problems the column by column approach allows for progressive updating, as in the optimization procedure described by Cornwell, Pegis, Rigler and Vogl (1973).

Finally, there are a number of places where, in the original algorithm, zero is included as a factor in multiplication. Modifications, including in some instances shortening of loops, have been applied to skip floating point multiplication where the result would be zero.

Application

The advantage of the generalized inverse is that it is unique and always exists, even in cases where a square matrix degenerates into singularity. In all cases the solution vector x = A+b is the vector of least Euclidean length which minimizes the Least Squares sum || A.x-b ||2. For further information on the generalized inverse see Penrose (1955), Peters and Wilkinson (1970), Greville (1959), Price (1964), Yassa (1974), Osborne (1965) and Bjerhammer, (1973).

In the basic GINV algorithm it is necessary that M > N. However by using the identity

\[ A^+ = A^T + \] (2)

it is possible to use MXTRAP to first produce A+T(N,M), wherein N>M, call GINV with N preceding M in the parameter list to produce A+T = A+T, and finally call MXTRAP to produce A+. Also it is possible to modify the algorithm so that this type of proceeding occurs automatically irrespective of whether M ≥ N. A further alternative is to remove the statement CALL MXTRAP (A,M,N) from the GINV subroutine and use an appropriate
CALL MXTRAP (A,M,N) statement before or after calling GINV with the M and N in correct sequence with the larger one entered first. For simplicity, the algorithm as presented here has been set up for the most usual case where $M > N$, with a call to MXTRAP included in GINV.

The dimensions of $A$, $U$, $AT$, $NF$, $C$, $X$ and $CQ$ must be at least of dimensions $N-1$, $N-1$, $M$, $N$, $N$ respectively.

The generalized inverse $A^+$ (M,M) satisfies, and in fact can be defined by, Penrose's Lemmas:

\[
\begin{align*}
A.A^+A & = A \\
A^+A & = A^+ \\
A^+ & = VQ^+U^T
\end{align*}
\]

These Lemmas may thus be used as a convenient basis for checking the numerical accuracy of the algorithm. Evidently the Cayley inverse of a non-singular square matrix also satisfies these Lemmas.

There may be applications where the factorization to $A.Z = Q$ would be useful. For the general case $Z$ is upper triangular and $Q$ is orthonormal in as many columns as the rank of the matrix, for the case $M > N$. As stated previously this factorization is used in the GINV algorithm. The additional subroutine GSM calls GINV in such a way that an exit occurs immediately after the factorization, at which point $A$ will have been overwritten by $Q(N,M)$ and $U$ by $Z$.

It should be noted that in both direct and indirect applications of GINV $A$ will have been overwritten by $A^+$ or by $Q$ respectively. Hence, if $A$ is required for later applications, its contents should be stored in another array before a call is made to either GINV or GSM.

### Tests of the Algorithm

The subroutine GINV has been used with fully satisfactory results on a number of different test matrices including the three $A_1$, $A_2$ and $A_3$ used by way of illustration below. $A_1$ is a $4 \times 4$ ill-conditioned matrix of full rank given by Cornwell (1973), $A_2$ is an $8 \times 5$ matrix of rank 3 given by Golub and Reinsch (1970) and $A_3$ is a $10 \times 6$ matrix of rank 3 given by Chen (1973). One term only of $A_3$ differs by one least significant digit in the last place as given by Chen from $A_3^+$ as computed by GINV. Also the latter agrees within better than $7.42 \times 10^{-14}$ for all terms of $A^+$ separately calculated by the author using singular value decomposition of $A$ (Golub and Reinsch: 1970). Results for the test matrices $A_1$, $A_2$, and $A_3$, summarized in Table I, below, include information such as $M$, $N$, rank, condition number, time taken and checks of accuracy using a CDC 7600 computer and with $NB$ set equal to 47 on line 18 of GINV. Times given are based upon repeating $NT$ times the factorization to $A+$. For simplicity, the algorithm as presented includes the three $A_i$, $A_i^+$ separately calculated by the author using singular value decomposition of $A$ (Golub and Reinsch: 1970). Results for the test matrices $A_1$, $A_2$, and $A_3$, summarized in Table I, below, include information such as $M$, $N$, rank, condition number, time taken and checks of accuracy using a CDC 7600 computer and with $NB$ set equal to 47 on line 18 of GINV. Times given are based upon repeating $NT$ times the factorization to $A+$. For simplicity, the algorithm as presented includes the three $A_i$, $A_i^+$ separately calculated by the author using singular value decomposition of $A$ (Golub and Reinsch: 1970). Results for the test matrices $A_1$, $A_2$, and $A_3$, summarized in Table I, below, include information such as $M$, $N$, rank, condition number, time taken and checks of accuracy using a CDC 7600 computer and with $NB$ set equal to 47 on line 18 of GINV. Times given are based upon repeating $NT$ times the factorization to $A+$.
Forran", Laboratory Note SN/54, 11/08/1976, CSIRO, Division of Textile Physics.


1. INTRODUCTION

This paper describes a scheme which has been developed for recording digital information on ordinary audio tapes with an unmodified audio cassette recorder. Using a single track on a recorder with a maximum frequency response of about 12.5 KHz a bit density of 2400 bpi has been achieved with very reliable operation. The equivalent character rate is 500 9-bit characters per second.

The system actually in use records on both channels of a stereo recorder to obtain a character rate of 1000 9-bit characters per second.

The stereo system is at present interfaced to the HP2100A minicomputer in the Department of Computer Science at Monash University. It is set up as an archiving device and is being used, especially by students, for archiving programs and data files from disk.

As the cassette recorder has not been modified in any way — the only connection to it is an audio lead — all tape motion must be controlled manually by pressing the appropriate buttons on the recorder. This is no disadvantage for simple archiving. The result is an economical device with quite a respectable transfer rate.

2. THE MODULATION SCHEME

The audio signal comprises a sequence of segments of each of three different frequencies $f_u$, $f_c$, and $f_l$, where $f_u > f_c > f_l$. There is always one frequency present and the value of the next bit is determined by which of the remaining two frequencies the signal switches to next. Thus one bit is recorded for each frequency transition and the signal is completely self clocking.

This scheme can of course be extended to more than three frequencies. If there are $m$ frequencies then there are $(m-1)!$ choices for the next frequency. Each transition can thus be used to record $(m-1)$ different symbols. The simplest system is with $m=3$ which is the reason a three frequency system was chosen.

Even with a fixed value of $m$ there are a number of different encoding schemes possible. For each frequency, symbols have to be assigned to each of the possible $(m-1)$ frequency transitions. This can be done in $(m-1)!$ ways. Thus there are $(m-1)!$ $m$ different encoding schemes possible. Some of these schemes have useful properties which may aid their encoding and decoding.

The encoding scheme actually used is of a type where, for the $m$ frequency case, $(m-1)$ of the frequencies are selected and each of them is always associated with the same symbol regardless of the frequency from which the transition was made. The $m$th frequency is common and is used as the next frequency when the present frequency is that for the symbol $i$ and the next symbol is also $i$. Hence the self clocking property is always preserved.

In the three frequency scheme, $f_u$ is assigned to a one bit and $f_c$ to a zero bit. $f_l$ is the common frequency. The scheme is depicted in Figure 1.

The above allocation of frequencies is arbitrary except that $f_u$ is deliberately chosen to represent a zero. By recording eight bit characters together with a ninth even parity bit, each resulting nine bit character must contain at least one zero and hence visit $f_c$ at least once. This enables the demodulation electronics to track $f_c$ and hence compensate for variations in tape speed.

The audio signal is conveniently generated by a voltage controlled oscillator (VCO) whose control voltage is switched to obtain the correct sequence of frequencies.

\[ f_u = \text{"1"}, \quad f_c = \text{"0"}, \quad f_l = \text{common} \]

\[ \text{transitions from } f_u \quad \text{transitions from } f_c \quad \text{transitions from } f_l \]

Figure 1: The Meaning of each Frequency Transition.

*Senior Lecturer, Department of Computer Science, Monash University, Clayton, Victoria. Manuscript received 15th June 1976 and in revised form 21st June 1977. † The device described herein is covered by Patent Appl.
Digital Recording on an Audio Recorder

3. DEMODULATION

The critical part of the demodulation process is to detect in as few cycles as possible which of the three frequencies is present. A phase locked loop (PLL) has been used as the heart of the demodulator, especially because phase locked loops are now available quite cheaply as integrated circuits.

The PLL is connected as an FM detector in which the voltage controlled oscillator (VCO) tracks the frequency of the tape recorder audio output. The VCO control voltage, when suitably filtered, provides a voltage proportional to frequency. This voltage is fed to a pair of analogue comparators with digital outputs. One output (FU) detects the upper frequency \( f_u \) and the other (FL) detects the lowest frequency \( f_l \). The block diagram of the demodulation system is shown in Figure 3, together with examples of the various voltage waveforms in Figure 4.

It is possible to use other schemes for detecting the frequencies. For instance Gillis (1974) described a recording scheme in which a set of matched filters was used to detect one frequency out of a set of eight; the individual frequencies differed by integral multiples of the lowest frequency. Even with frequency doubling a period equal to two cycles of the lowest frequency was required for successful detection. By using a PLL it has been possible to detect a frequency reliably in about 1½ cycles of the lowest frequency. Also, the frequencies can be spaced more closely than integral multiples of the lowest frequency.

For fast response the internal low pass filter in the PLL must have a small time constant. Thus the VCO control voltage will contain a large double frequency component. Conventionally, the raw control voltage is subtracted from the control voltage at the VCO centre frequency and the difference passed through an external low pass filter. For a symmetrical lock range the VCO centre frequency has been chosen to be \( f_c \).

Both the frequency response and transient response of the external low pass filter are critical in determining the maximum bit rate which can be recovered. The filter should have a short rise time but should reject the lowest unwanted component at \( 2f_c \). Although a small amount of overshoot in the step response improves the rise time, too much may cause a transition from, say, \( f_u \) to \( f_c \) to erroneously enter the detection region of \( f_l \).

The filter used at present is a ninth order “damped Butterworth”. The poles are equally spaced on a circle as for a Butterworth filter, but instead of spanning over the full semicircle they have been restricted to the sector with an included angle of \( \frac{\pi}{2} \). The resultant filter has a less sharp frequency response but an improved transient response with less overshoot than the conventional Butterworth filter. Another possibility is a transitional Butterworth-Thomson filter. (Peters & Murakami, 1957.) However the “damped Butterworth” filter has the advantage that the poles do lie on a circle.

The output of the filter is a voltage proportional to frequency with \( f_u \) nominally at zero volts. This signal is converted to two booleans, FU and FL, by comparing it with two reference voltages equal and opposite about zero volts. FU and FL are in turn processed to determine the frequency transitions and hence the bits.

In determining the frequency transitions an ambiguity occurs between a single transition between the outer frequencies \( f_l \) and \( f_u \), and a double transition between \( f_l \) and \( f_u \) via the centre frequency. To resolve this, \( FU = FL \), which changes every time \( FU \) or \( FL \) changes, is full wave differentiated and used to trigger a non-retriggerable mono.

The pulse length of this mono is chosen to absorb the second of the pair of close trigger pulses which occur with the single transition between \( f_u \) and \( f_l \). The time between these pulses depends on the rise time of the filter and the spacing of the comparator threshold, and is fairly small compared with a normal bit time. Finally, the trailing edge of the mono is differentiated to obtain a bit clock. The corresponding bit value is just

\[ FU \lor FL, FU' \]

where \( FU' \) is the previous value of \( FU \).

The comparator thresholds and the mono pulse duration must both be adjusted to obtain maximum reliability. To appreciate the problem, consider the two situations in Figure 5. Situation (1) shows the trigger pulses in the case of a single frequency transition between the two outer frequencies, and (2) shows the case of a pair of frequency transitions.

For maximum tolerance the mono period should be chosen so that \( t_1 = t_2 \). However, as the pulse positions are subject to a fair amount of jitter due to noise and ripple...
Figure 3: Block Diagram of the Demodulator.
going through the filter, a better requirement is $t_1 > t_{m1}$ and $t_2 > t_{m2}$, where $t_{m1}$ and $t_{m2}$ are minimum safe values; for simplicity they can be made equal.

Adjustment of the mono pulse length can be made using an oscilloscope, but to make the job easier two lights have been provided in the circuitry. One is flashed if, in situation (1) Figure 5, the second trigger pulse arrives less than $t_{m1}$ before the mono times out. The other light is pulsed if, in situation (2), the second trigger pulse arrives less than $t_{m2}$ after the mono times out. The period of the mono can now be optimised simply by adjusting it until both lights are always on or both flash equally. The times $t_{m1}$ and $t_{m2}$ are set by the periods of two further monos.

Altering the comparator thresholds has the effect of altering the spacing of the trigger pulses. For maximum tolerance the time $t_1 + t_2$ should be maximised. This can be shown to occur when the thresholds are set equal to $\frac{3}{\pi} V_p$, where $V_p$ is the peak voltage of the filter output.

Of course it would also be possible to derive a bit clock at frequency transitions by detecting zeros in the analogue derivative of the filtered VCO control voltage. However, the problem is to ensure that a well defined zero derivative occurs at $f_c$ when two successive transitions have the same signed derivative. The digital approach taken is sensitive only to the actual rise time of the filtered VCO voltage and does not rely on well defined zero derivatives.

4. FREQUENCY TRACKING

In practice the frequencies as played back will be subject to some variation. The main causes are tape speed fluctuations and drifts in the modulator oscillator. Drifts in the modulator and/or the comparators the tolerance of double frequency ripple can be increased to a certain extent.

On the other hand, if the tape speed increases, the bit rate increases and the time spent at each frequency decreases. Thus the filter cutoff frequency, its rise time, the bit rate, and the amount of speed variation which can be tolerated are all a matter of compromise.

The final consideration is to ensure that the tracking scheme will initially lock in. Because of capacitive coupling in the DC restoration circuit, the output will decay to zero whenever the frequency is constant for long. As zero output is interpreted as $f_c$, it is only necessary to ensure that $f_c$ is present on tape under the "no signal" condition, i.e. on the tape leader and in interblock gaps. This is achieved simply by designing the modulator to reset to $f_c$ if no bit has been recorded for some time. Note that apart from this requirement that $f_c$ be present initially and in gaps, there is no need for any special preamble to be recorded.

6. DETECTION OF END OF CHARACTER AND END OF BLOCK

The present version has no way of determining the end of a character except by counting nine bits. This means that if a bit is lost (say) all following bits until the end of the block will usually be in error. If end of character is detected by additional information such propagation of errors is eliminated.

Detection of end of character is easily provided for by a pause between the last bit of one character and the first bit of the next character. End of character can be detected when a mono, which is retriggered by each frequency transition, times out. The end of character pause must be significantly longer than any pause to aid frequency tracking but still need not be much longer than a bit time. Provision for detecting end of character by a pause, of course, degrades the average bit rate but it does stop errors propagating.

In fact end of block is actually detected by a pause of minimum length equal to about one character time.

7. OPERATION AS A COMPUTER PERIPHERAL

Because all tape motion must be initiated manually it is necessary to ensure correct co-ordination between the operator and the archiving program. This is really quite simple.

The interface contains a READY flip-flop which can be cleared by the program and set by pressing a push button. There are also two lights, RECORD and PLAYBACK, which can be illuminated under program control. When the program is ready to read/write it clears the READY flip-flop and illuminates the PLAYBACK/RECORD light. The operator then starts the tape recorder in the required mode and, when the tape leader has passed, presses the push button. This sets the READY flip-flop which in turn causes an interrupt in the computer; alternatively the state of the READY flip-flop can be tested by a status check on the interface. Once the interface is ready the program commences the transfer.

The cassette recorder presently in use does not have separate read and write heads. Thus simultaneous reading and writing is not possible for a read after write check. However, the archiving program has been written so that the tape can be completely reread after it has been written and the information which is read back checked against that which was used to write the tape. If a hard error is detected the tape has to be completely rewritten.
Figure 4: Wave Forms in Demodulator.
However, provision has been made for rereading blocks on tape which contain sum check or parity errors. All physical blocks are prefixed by a sequential block number which is included by software as an extra pair of characters before the rest of the information in the block. Also, the interface is capable of reading until the end of a block of unknown length. Thus, if reading is commenced from an arbitrary point on tape within valid information, the program can determine the location within the file.

Now, if an error is detected during normal reading, the program displays a message requesting a reread and resets the READY flip-flop. The operator rewind the tape a short distance by pressing fast reverse, presses playback and then the READY button. The program then starts reading again until the required block in error is reached. Any number of block rereads can be performed in this way. The same procedure, but using fast forward, can be used to skip to information well up the tape when earlier information is not required to be read in this particular run.

8. RELIABILITY

Systematic testing is a very laborious process especially with the present system owing to its manual initialisation and moderate data transfer rate. For example it would take approximately three hours to transfer $10^8$ bits!
In fact, a series of reliability tests were carried out in which fairly new SONY C30 and SONY C60 low noise tapes were each recorded with 1000 blocks of 256 characters. The tapes were then each read back six times resulting in only one soft error; the first reread over this error was successful. The total number of bits transferred was $3 \times 10^7$.

As tests of the above type are far from reality, in which there are a number of users (mainly students) using an assortment of tapes, the system was made available on an experimental basis. In fact it turned out to be very reliable. The only troubles experienced were due to dirty heads, damaged tapes and some difficulty with their long play (C120) tapes.

Firstly, it was found necessary to clean the heads every few weeks. Secondly some users bought really old tapes from home and these were found to be quite visibly wrinkled in places resulting in severe dropouts. Thirdly, two fairly low quality new C120 tapes initially gave hard errors. These were diagnosed as being due to the tape stretching in spots and producing unusually long cycles of audio which in turn caused the phase locked loop to momentarily lose lock.

Actually C30 tapes (15 minutes per side) are plenty long enough for most purposes as they have a capacity of 900,000 characters per side.

Overall, in one year of use, apart from the above problems, only one hard error was experienced and this was attributed to a small visibly missing chip of oxide on a new tape. Very occasionally a soft error occurred but with the reread facility, described in the previous section, little extra delay was caused. The stability of information is good and I have files recorded over a year ago which are still read error free.

9. DISCUSSION

There have been a few other systems described for recording digital information on audio recorders. However, these tend to use either conventional digital techniques such as phase encoding and two-frequency FM (Newton and Buczek, 1970) or simple two frequency FSK schemes (Chin, 1973; Rowe, 1975).

By using more than two frequencies and using only one frequency transition per bit the present scheme is able to achieve reliable operation at a very respectable bit density. Also, the frequencies do not have to be simple ratios of each other nor do they have to be phase locked to the bit rate. This allows each tuning to obtain reliable operation with almost any recorder.

In fact the system has been used with other tape recorders, two of which were only designed for speech reproduction. By decreasing the three frequencies, whilst maintaining their relative spacing, and reducing the cutoff frequency of the low pass filter, reliable operation was achieved. For example, using an OPTACORD 448 recorder which has a frequency response 3dB down at 6KHz with 5mV input, reliable operation was achieved at 330 cps on a single track. The corresponding bit rate is 1600 bpi.

Summarising, the advantages of the scheme described here are:

1. it is completely self clocking,
2. it is insensitive to fairly large tape speed fluctuations,
3. the bit rate can easily be traded for reliability,
4. end of character and end of block are easily detected,
5. there is no need for any special preamble or postamble.

References


168 The Australian Computer Journal, Vol. 9, No. 4, November, 1977
Forthcoming National and International Events

<table>
<thead>
<tr>
<th>November/December</th>
<th>Event Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 Nov - 2 Dec</td>
<td>1977 SIGMETRICS/CMG VIII: An International Conference on Computer Performance</td>
<td>Washington, DC, USA</td>
</tr>
<tr>
<td>5 - 7 Dec</td>
<td>Winter Simulation Conference</td>
<td>Gaithersburg, Maryland, USA</td>
</tr>
<tr>
<td>5 - 9 Dec</td>
<td>Third International Symposium on Computing Methods in Applied Sciences and Engineering</td>
<td>Rocquencourt, France</td>
</tr>
<tr>
<td>5 - 9 Dec</td>
<td>Third International Colloquium on Methods of Scientific and Technical Calculation</td>
<td>Versailles, France</td>
</tr>
<tr>
<td>15 Dec</td>
<td>Computer Networking Symposium</td>
<td>Gaithersburg, Maryland, USA</td>
</tr>
</tbody>
</table>

1978

<table>
<thead>
<tr>
<th>January</th>
<th>Event Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>23 - 25 Jan</td>
<td>Fifth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages</td>
<td>Tuscon, Arizona, USA</td>
</tr>
<tr>
<td>23 - 27 Jan</td>
<td>Fifth Panel Discussion on Computation Topics</td>
<td>Valparaiso, Chile</td>
</tr>
<tr>
<td>February</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>13 - 15 Feb</td>
<td>Symposium on Computer Network Protocols</td>
<td>Liege, Belgium</td>
</tr>
<tr>
<td>23 - 24 Feb</td>
<td>Australian Universities Computer Science Seminar 1978</td>
<td>NSW, Australia</td>
</tr>
<tr>
<td>23 - 24 Feb</td>
<td>SIGCSE/CSA Symposium</td>
<td>Detroit, Michigan, USA</td>
</tr>
<tr>
<td>March</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>15 - 17 Mar</td>
<td>Eleventh Annual Simulation Symposium</td>
<td>Tampa, Florida, USA</td>
</tr>
<tr>
<td>May</td>
<td>Conference on Computers in Banking and Finance</td>
<td>Bled, Yugoslavia</td>
</tr>
<tr>
<td>22 - 25 May</td>
<td>Sixth International CODATA Conference</td>
<td>Taormina, Sicily</td>
</tr>
<tr>
<td>31 May - 2 June</td>
<td>1978 ACM-SIGMOD International Conference on Management of Data</td>
<td>Austin Texas, USA</td>
</tr>
<tr>
<td>June</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>12 - 16 Jun</td>
<td>Seventh Triennial World Congress</td>
<td>Helsinki, Finland</td>
</tr>
<tr>
<td>August</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>6 - 8 Aug</td>
<td>Jerusalem Conference on Information Technology</td>
<td>Jerusalem, Israel</td>
</tr>
<tr>
<td>21 - 25 Aug</td>
<td>COMPSTAT 1978</td>
<td>Leiden, The Netherlands</td>
</tr>
<tr>
<td>August/September</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>28 Aug - 1 Sept</td>
<td>8th Australian Computer Conference</td>
<td>Canberra, Australia</td>
</tr>
<tr>
<td>September</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>4 - 8 Sept</td>
<td>Third International Congress on Electronic Information Processing (IKD)</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>4 - 8 Sept</td>
<td>Medical Informatics Europe: First Congress of the European Federation for Medical Informatics</td>
<td>Cambridge, UK</td>
</tr>
<tr>
<td></td>
<td>South East Asia Regional Computer Conference 1978 (SEARCC 78)</td>
<td>Manila, The Philippines</td>
</tr>
<tr>
<td>October</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>15 - 19 Oct</td>
<td>American Society for Information Science Annual Meeting</td>
<td>New York, USA</td>
</tr>
<tr>
<td></td>
<td>WC Network Interconnection</td>
<td>Tokyo, Japan</td>
</tr>
<tr>
<td>November</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td></td>
<td>International Working Group Meeting Computer Applications in Engineering Design</td>
<td>Latin America</td>
</tr>
<tr>
<td>1979</td>
<td>Event Description</td>
<td>Location</td>
</tr>
<tr>
<td>June</td>
<td>IFAC/IFIP 2nd International Symposium, Software for Computer Control</td>
<td>Prague, CSSR</td>
</tr>
<tr>
<td></td>
<td>IFIP/IFAC Symposium, Hardware for Computer Control</td>
<td>Zurich, Switzerland</td>
</tr>
</tbody>
</table>

The Australian Computer Journal, Vol. 9, No. 4, November, 1977
THE FIRST

World Computer Conference

To ever be held in Australia
will be in Melbourne OCTOBER 14 - 17, 1980

IFIP CONGRESS 80
International Federation for Information Processing

will be organized and managed by creative, dynamic people from Australian Data Processing Community ... You could be one of these people ... 

Drop a line to the Australian Organizing Committee
IFIP CONGRESS 80
P.O. BOX 880G,
Melbourne, 3001

If you have the time and talents in any of the following areas and really want to be part of the team ...

- Conference Facility Organization
- Audio Visual
- Secretarial
- Equipment Exhibition Organization
- Publicity/Public Relations
- Technical Writing
- Social Event Organization
- Special Computer Orientated Event Organization i.e. Chess, Art, Music, etc.
- Information Distribution/-Mailing Lists
- Conference Newspaper Production/Editorial.

For further information contact the IFIP Correspondent, A. Y. Montgomery, Department of Computer Science, Monash University, Clayton, Victoria, 3168.
Presenting lunch hour programming.

Programming on other computers can eat up a lot of valuable time. But on Data General commercial ECLIPSE systems you can develop workable screen formats in as little time as an hour. So instead of having to wait for a break in your computer's schedule, you could do a small data entry program on your lunch break. And wrap up entire applications in days rather than months.

A Data General commercial ECLIPSE data processing system can do it because it's a real-time data management system. It uses Idea, our interactive data entry/access software for online screen formatting. Idea also has an online multiterminal monitor and powerful English-like language resulting in faster screen-related programming. Plus applications developed under Idea can automatically access and maintain files built by a highly sophisticated data management system, our INFOS software.

Our commercial ECLIPSE data processing systems also use ANSI '74 COBOL implemented at the highest levels. For fast batch program execution with minimal programmer retraining. Upgrading our system is no big deal either. It can grow right along with your expanding processing appetite.

Commercial ECLIPSE systems. The computer systems that digest programming easily and quickly. Send for our booklet today. Time is nibbling away.

Mail to: Data General Australia Pty. Ltd.,

☐ Send me your commercial ECLIPSE brochure.
☐ Send me your ECLIPSE brochure and have sales representative call.

Name________________________________________________________
Title________________________________________________________
Company_______________________________________________________
Address ____________________________ _________________________
P-code________Tel.________

ECLIPSE is a registered trademark of Data General Corporation © Data General Corporation, 1977.
Data General Australia Pty. Ltd., 96 Camberwell Road, Hawthorn East, Victoria, 3123. Tel. 82 1361, Sydney 908 1366, Adelaide 223 7342, Brisbane 29 5744, Perth 21 5981, Data General New Zealand Ltd., Wellington 723 095, Auckland 378 738.

Data General
It's smart business.
Contents:

132-137 An Integrated System for Administration of Managed Funds
By C. PRIME

138-144 The Implementation of a Data Base Management System
by H.G. MACKENZIE and J.L. SMITH

145-149 A Geographic Data Base Using FORDATA
By G.C. COOK and K.K. MAYO

150-154 Development of a Bibliographic Search System
Using FORDATA
By J.D. SHORTRIDGE

155-158 A Query/Update Package for Library or Personal Use
By H. MACKENZIE and G. KELLY

159-161 GINV, A Subroutine in ANSI Fortran for Generalized Matrix Inversion
By H.W. HOLDAWAY

162-168 High Density Digital Recording on an Audio Recorder
By D.M. BOULTON

Special Features:

144, 154 Book Reviews

169 Forthcoming National and International Events

170 IFIP Congress 80

REPRINTS: 50 copies of reprints will be provided to authors. Additional reprints can be obtained, according to the scale of charges supplied by the publishers with proofs. Reprints of individual papers may be purchased for 50 cents each from the Printers (Publicity Press).

PAPERS: Papers should be submitted to the Editor; authors should consult the notes published in Volume 9, pp. 45-49 (or request a copy from the National Secretariat).

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous in local matters, and may charge different membership subscriptions. Information may be obtained from the following Branch Honorary Secretaries. Canberra: P.O. Box 446, Canberra City, A.C.T. 2601. N.S.W.: P.O. Box N250, Grosvenor St, Sydney, N.S.W. 2000. Qld: Box 1484, G.P.O., Brisbane, Qld. 4001. S.A.: Box 2443, G.P.O., Adelaide, S.A. 5001. W.A.: Box F320, G.P.O., Perth, W.A. 6001. Vic.: P.O. Box 98, East Melbourne, Vic. 3002. Tas.: P.O. Box 216, Sandy Bay, Tas. 7005.