DOCUMENT RESUME

ED 074 766		EM 010 964

TITLE	APL/IV: Fourth International APL Users' Conference. June 15-16, 1972, Atlanta, Georgia, U.S.A.
INSTITUTION	Atlanta Public Schools, Ga. Computer Center.; Georgia Inst. of Tech., Atlanta. School of Information Science.
SPONS AGENCY	Atlanta Board of Education, Ga.
PUB DATE	Jun 72
NOTE	194p.
EDRS PRICE	MF-$0.65 HC-$6.58
DESCRIPTORS	*Bibliographies; *Computer Assisted Instruction; Computer Graphics; *Computer Science; *Conference Reports; *Programing Languages; Statistical Analysis; Time Sharing
IDENTIFIERS	APL; *A Programming Language

ABSTRACT

APL is a computer language (A Programing Language). Papers at this conference of APL users deal with the following topics: an APL approach to interactive display terminals; graphics in APL; an interactive APL graphics system; modeling a satellite experiment on APL; representing negative integers in bit vectors; APL as a teaching tool--two versatile tutorial approaches; the evolution of an interactive chemistry laboratory program; a collection of graph analysis APL functions; management of APL time-sharing activities; saving money by saving space in APL; security of APL application packages; enhanced interaction for an APL system; subtasking in APL; suggestions for a "mapped" extension of APL; APL as a notation for statistical analysis; an adaptive query system; microprogram training--an APL extension; and APL electronic circuit analysis program and use of APL in teaching electrical network theory. Also included is a bibliography of 340 items dealing with APL. (JK)

ED 074766

APL\IV

FOURTH INTERNATIONAL APL USERS' CONFERENCE

JUNE 15-16, 1972

ATLANTA, GEORGIA U.S.A.

PROCEEDINGS

OF THE

FOURTH INTERNATIONAL

APL USERS' CONFERENCE

JUNE 15 - 16, 1972

ATLANTA GEORGIA U.S.A.

Special thanks are due to Bernard McIlhany who supervised the production of these proceedings; to Linda Reagan, Gloria Quattlebaum, Betty Black and Bobbie Wingo who entered all of the papers into the ATS terminal system; to Eddie Peabody who did all the layout work; to Mike Massey and Bill Maury who made numerous corrections and suggestions of a technical nature; to the entire operations department of the Atlanta Public Schools' Computer Center who helped again and again with proof production; and to the staff of McDaniel Printing Company.

TABLE OF CONTENTS

FOREWORD

The Fourth International APL Users' Conference was held in Atlanta, Georgia, on June 15 and 16, 1972. The conference was co-hosted by the Computer Center, Atlanta Public Schools, and the School of Information and Computer Science, Georgia Institute of Technology.

The program was arranged by Dr. Garth Foster, Syracuse University, Syracuse, New York. Dr. Foster was responsible for the refereeing of the papers and the establishment of the fine program.

These proceedings were compiled at the Atlanta Public School Systems' Computer Center using ATS/360. Editing was accomplished by staff members from the Computer Center and the Computer Braille Project. The print shop of the Atlanta Area Technical School was most cooperative in providing printing services for conference brochures.

The local arrangements were the responsibility of Ms. Jackie Reynolds, Systems Analyst at the Computer Center. Ms. Reynolds handled all registrations and hotel accommodations for the conference and has contributed many hours of her personal time to ensure the success of the conference.

It is obvious that a conference such as this would not be successful without the contributions made by the speakers. They have contributed greatly to the sessions and to the overall efforts toward the proliferation of APL in the computer community.

As conference attendees, you are all to be commended for making this conference a success. APL, as is brought out in one of the papers, is epidemic and you are all contributors to the cause.

There are too many to individually cite, who performed hours of thankless chores for this conference. Many letters had to be typed and mailed and many telephone calls had to be answered. It quite obviously is a great effort to host a conference such as this. I would like to take this opportunity to collectively thank everyone who helped to make this conference a success.

Thomas J. McConnell, Jr.
Arrangements Chairman

IS APL EPIDEMIC ? OR A STUDY OF ITS GROWTH THROUGH AN EXTENDED BIBLIOGRAPHY

J. C. Rault and G. Demars
Laboratoire Central de Recherches
THOMSON-CSF
Domaine de Corbeville, B. P. 10
(91) Orsay, France

Summary:

An attempt is made to demonstrate that the use of APL is growing in an epidemic fashion. The theory of epidemic processes is applied in an approximate manner by means of data provided by a nearly-exhaustive bibliography given as an appendix. APL is proved to be undoubtedly epidemic.

A second part details the given bibliography.

Introduction

The few conferences held these past years on APL have demonstrated a fast growth of its use. At this time APL, as we hope this conference will show, is pervading every area of activity.

To the dismay of the vilifiers of APL, this pattern of development resembles the spread of an infectious disease.

The purpose of this paper is twofold: first establishing that APL is an epidemic, a commonplace assertion among APL supporters; second providing the APL community and the APL addicts to-come with an extended bibliography given in appendix. To our knowledge, such a bibliography has not been made available so far.

The Epidemiology of APL

The spread of scientific ideas has already been studied in terms of an epidemic process[1, 2, 3]. A thorough theory has been laid down and applied to the study of an entire discipline such as symbolic logic[3] for a one-hundred year time interval.

Along these lines we attempt here to apply the same theory to the growth of APL using as a data base a bibliography recently compiled.

Figure 1

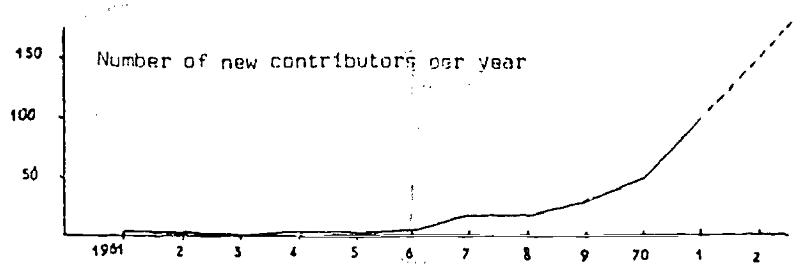


Figure 2

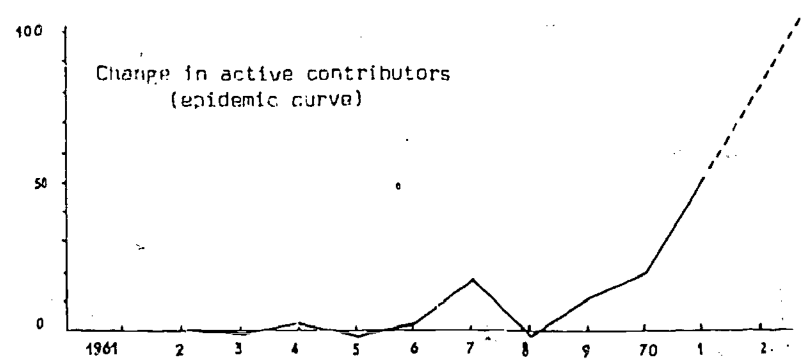


Figure 3

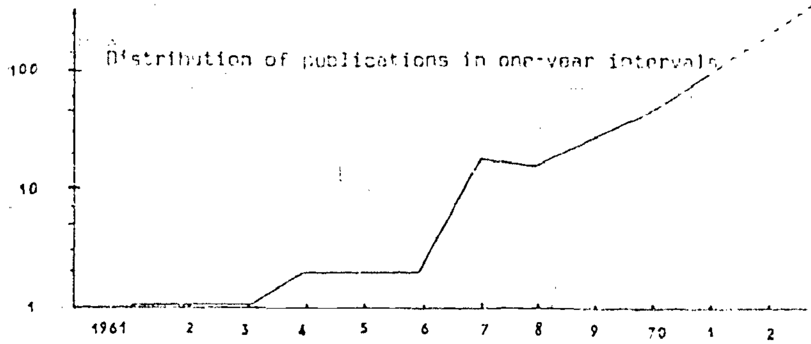
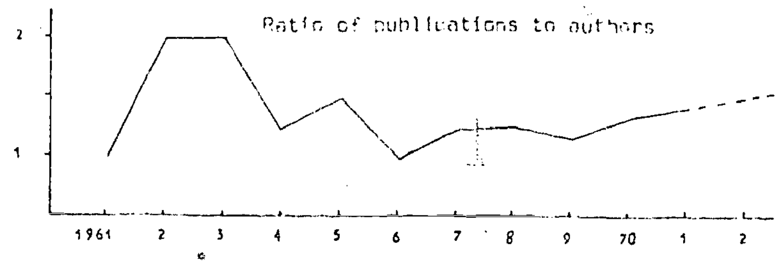


Figure 4



It would not be contended that this literature search is exhaustive and we are fully aware that improvements can be made.

This bibliography amounts to about 330 entries and 220 authors. Thus the APL epidemic process, investigated here, considers a population of 220 individuals, or infections, over a ten-year span. Taking into account that an entry may include several authors the total number of publications is 422.

Figure 1 shows the number of new contributors each year; figure 2, showing the change in the number of active contributors each year, represents the epidemic curve for APL since its inception. This curve reveals clearly that since 1968 APL is really an epidemic. This corresponds to the release of an "infectious material" by IBM, APL/360, as a class III product.

The shape of this epidemic curve does not allow one to foresee that the epidemic will stabilize in the near future, which, according to theory, should occur when the curve ceases to increase. (In fact this curve tends to grow exponentially).

Figure 3 gives the yearly number of publications; if the present rate of growth is maintained this year we may expect 250 papers in 1972 with 40 new authors.

Figure 4 indicates the yearly average number of publications per author. This ratio has increased continuously since 1969.

The above figures should not be taken as accurate ones but just as mere benchmarks manifesting that APL, in spite of its infectious character to certain people, is hale and hearty and thriving at a pace which may endanger soon the bailiwicks of those die-hard fossils that FORTRAN, BASIC and other patters are.

An Annotated Bibliography

Perusing the appended bibliography is sufficient to be convinced that APL is present in many fields. We intend here to make general comments for facilitating the use of this bibliography.

APL Implementations One may note the fact, which is not always well known except to specialists, that APL may now be found on major computers outside of IBM:

BURROUGHS:	23, 98, 144-46, 247, 298
CDC:	58
CII:	157-8, 181-2, 195-6, 198
DEC:	216
HONEYWELL/GE:	95
IBM:	22, 59, 77-80, 82, 312, 331
UNIVAC:	339
XDS:	21, 242-43, 285, 340

[Ed. note: numbers here refer to bibliography, not hardware!]

Microprogrammed APL machines: 16, 45, 102-106, 203, 207, 295-297

APL Compatible Terminals

A wide range of APL compatible peripherals are available. Reference 177 gives a nearly complete list of them (48, 62, 86, 101, 188, 250, 307).

APL File Handling Capabilities

Users of APL quite early have demanded facilities to work with large collections of data under program control. A number of file systems have been experiemnted or are presently available (37, 63, 84, 179, 189, 202, 262, 334, 338).

APL Handbooks

Many books, handbooks, user's manuals, videotapes and other materials are available: 6, 10, 23-24, 42, 43, 78-80, 95, 112, 140-2, 205, 219, 232, 235-38, 320-323, 329.

APL AS AN AID FOR DEVELOPING PROGRAMS: 8, 17
APL BATCH PROCESSING: 51, 289
APL DATA ACQUISITION: 149, 226
APL ENHANCEMENT AND EXTENSIONS: 36, 49, 50, 152, 170, 215, 248
APL GRAPHICS: 72, 150, 177, 235
APL HISTORY: 14, 261, 302
APL IMPLEMENTATION: 1, 2, 5, 16, 102-106, 156, 203, 207, 224, 231, 252-3, 287, 292
APL PLOTTING: 20, 62, 177, 239
APL SEMANTICS: 7, 230, 233, 245-46
APL THEORY: 32, 40, 47, 120, 159, 215, 230, 233, 245-46, 252-3, 279

In the following we give for the most significative fields of application, the appropriate references where the reader may find more detail:

COMMERCIAL: 20

COMPUTER AIDED INSTRUCTION: 26, 29, 56, 71, 83, 99-100, 111, 116, 127-8, 133, 165, 168, 176, 222, 254, 273, 304, 327

DATA BASES: 171-73

ENGINEERING:

- Digital Systems: 15, 29, 31, 33, 55, 61, 69, 74-76, 81, 89, 91, 110, 137-8, 160, 163, 199, 200, 206, 240
- Electrical Networks: 18, 19, 68, 97, 208-214, 218, 257, 259, 286
- General Engineering: 166, 191, 258
- Mechanical Design: 66, 139
- Survey: 174

FINANCE: 254

INSURANCE: 67, 135, 337

MANAGEMENT: 38, 87, 93, 162, 164, 255, 282, 291, 336

MATHEMATICS:

- Complex Arithmetic: 70
- Fast Fourier Transform: 139, 183, 280
- Formal Computation: 179, 280
- Graph Theory: 68
- Linear Programming: 194
- Numerical Analysis: 20, 27, 41, 46, 136, 276
- Operations Research: 112
- Optimization: 9
- PERT: 20, 192-93, 217, 265, 326, 331-32
- Sorting: 293
- Statistics and Probability: 11, 12, 20, 67, 108, 129, 155, 263-64, 266-71, 290
- Theorem Proving: 107, 204
- Walsh Functions: 251

MEDICINE: 300

RECREATION AND GAMES: 310, 311

TAXES: 260

Conclusions

Aside from the polemical aspect of this paper aimed to pique APL detractors, hope that it will be a contribution to the spreading of APL. We propose that this bibliography be augmented, improved and refined, possibly with the help of a KWIC index.

ACKNOWLEDGEMENTS

The authors wish to thank Dr. P. Abrams of CEGOS-INFORMATIQUE, Puteaux, France, and M. V. Chaptal of l'IRIA, Rocquencourt, France, for their contributions which aided in the preparation of this bibliography.

BIBLIOGRAPHY

1. W. Goffman. "Mathematical approach to the spread of scientific ideas", Nature, 212, pp. 449-452, October 1966.

2. W. Goffman and V. A. Newill. "Communications and epidemic processes", Proc. Royal Soc., A298, pp. 316-334, May 1967.

3. W. Goffman. "A mathematical method for analyzing the growth of a scientific discipline", Journal of the ACM, Vol. 18, No. 2, pp. 173-185, April 1971.

APPENDIX

(An APL Bibliography)

AN APL BIBLIOGRAPHY APRIL 1972

(1) P.S.ABRAMS=AN INTERPRETER FOR 'IVERSON NOTATION',TECH.REPT CS-47,COMPUTER SCIENCE DEPT.STANFORD UNIVERSITY,STANFORD CALIFORNIA,17 AOUT 1966.

(2) P.S.ABRAMS=AN APL MACHINE,PH.D. THESIS,STANFORD UNIVERSITY, STANFORD LINEAR ACCELERATOR CENTER,RPT NO.SLAC-114, FEVRIER 1970 ET AD.706-741.

(3) P.S.ABRAMS=INTRODUCTION AU LANGAGE APL,REVUE CEGOS-INFORMATIQUE,NO.34,P.5-7,MARS-AVRIL 1970.

(4) P.S.ABRAMS ET W.M.MC KEEMAN= COMPUTER DISPLAY OF THE DERIVED POLYTOPES,REVUE CEGOS-INFORMATIQUE,NO.36,P.25-36,JUILLET-AOUT 1970.

(5) P.S.ABRAMS=UNE NOUVELLE MACHINE POUR APL ,AFCET,CONGRES D'INFORMATIQUE,BROCHURE NO.2,P.85-106 ,PARIS,SEPTEMBRE 1970.

(6) P.S.ABRAMS ET G.LACOURLY=LE LANGAGE DE PROGRAMMATION APL, UNE INTRODUCTION,CEGOS-INFORMATIQUE 1971.

(7) P.S.ABRAMS=A FORMAL APPROACH TO APL SEMANTICS,COLLOQUE APL, 9-10 SEPTEMBRE 1971,PUBLICATION IRIA P.159-80.

(8) B.AMY,D.BASTIN,E.GIRARD ET J.C.RAULT=L'APL-UN OUTIL POUR LE DEVELOPEMENT DES PROGRAMMES,REVUE TECHNIQUE THOMSON-CSF, SEPTEMBRE 1972.

(9) B.AMY=A PROPOS D'UN PROGRAMME D'OPTIMISATION-ANALYSE NUMERIQUE ET LOGIQUE DES PROGRAMMES EN APL,REVUE TECHNIQUE THOMSON-CSF,SEPTEMBRE 1972.

(10) A.ANGER=THE APL LANGUAGE,J.WILEY,1971.

(11) F.J.ANSCOMBE=USE OF IVERSON'S LANGUAGE APL FOR STATISTICAL COMPUTING,DEPT OF STATISTICS,YALE UNIVERSITY,TECHNICAL REPT NO.4,JUILLET 1968 ET RAPPORT AD 672-557.

(12) F.J.ANSCOMBE=STATISTICAL COMPUTING WITH APL,1970.

(13) C.R.ATTANASIO,G.WALDBAUM ET F.ZARNFALLER=THE IMPLEMENTATION OF APL/360 FOR OPERATING SYSTEM/360,IBM RESEARCH DIVISION YORKTOWN HEIGHTS,N.Y.,4 JUIN 1968,RC-2109.

(14) J.N.BAIRSTOW=MR IVERSON'S LANGUAGE AND HOW IT GREW,COMPUTER DECISIONS,VOL.1,NO.1,P.42-45,SEPTEMBRE 1969.

(15) D.BASTIN,E.GIRARD ET J.C.RAULT=LA SIMULATION DES CIRCUITS LOGIQUES A L'AIDE D'UN SYSTEME APL,REVUE TECHNIQUE THOMSON-CSF,SEPTEMBRE 1972.

(16) G.BATTAREL,M.DELBREIL ET P.KALFON=UN INTERPRETEUR APL AVEC GENERATION ET REUTILISATION DE CODE MACHINE,COLLOQUE APL 9-10 SEPTEMBRE 1971,PUBLICATION IRIA P.383-402.

(17) R.BAYER=TOWARD COMPUTER-AIDED PRODUCTION OF SOFTWARE FOR MATHEMATICAL PROGRAMMING,IN MATHEMATICAL SOFTWARE,J.R.RICE, ACADEMIC PRESS 1971,P.275-93.

(18) W.R.BEAM=AN APL IMPLEMENTATION OF MICROWAVE CIRCUIT ANALYSIS TECHNICAL APPLICATIONS PAPERS,NEREM 1970,P.99-105.

(19) R.BEAUFILS,P.CAZAUX ET M.LABORIE=APPLICATION D'APL A L'UNIVERSITE DE TOULOUSE,COLLOQUE APL,9-10 SEPTEMBRE 1971 ,PUBLICATION IRIA.

(20) J.BECKER=EDITPAK,FINANCEPAK,MATHPAK,PLOTPAK,STATPAK, SCIENTIFIC TIME-SHARING CORP.,1969.

(21) G.A.BERGES ET F.W.RUST=APL/MSU REFERENCE MANUAL,DEPT OF EE, MONTANA STATE UNIVERSITY,BOZEMAN,MONTANA,26 SEPTEMBRE 1968.

(22) P.C.BERRY=APL/1130 PRIMER,IBM CORP.,1968,FORM NO.GC-20-1697-0.

(23) P.C.BERRY=APL/360 PRIMER,STUDENT TEXT,IBM CORP 1969, FORM NO.GC 20-1702-0.

(24) P.C.BERRY=APL/360 PRIMER,IBM CORP.,FORM NO.GH-20-0689-1 2NDE EDITION JANVIER 1970.

(25) P.C.BERRY,A.D.FALKOFF ET K.E.IVERSON=USING THE COMPUTER TO COMPUTE,A DIRECT BUT NEGLECTED APPROACH TO TEACHING MATHEMATICS,NEW-YORK IBM SCIENTIFIC CENTER,REPORT NO.320-2988,MAI 1970 ,ET IFIP WORLD CONFERENCE ON COMPUTER EDUCATION,AMSTERDAM,24-28 AOUT 1970.

(26) P.C.BERRY,G.BARTOLI,C.DELL'AQUILA ET V.N.SPADAVECCHIA=APL AND INSIGHT,A STRATEGY FOR TEACHING,COLLOQUE APL,9-10 SEPTEMBRE 1971,PUBLICATION IRIA P.251-72.

(27) T.A.BICKART=FUNCTION TO ACCELERATE AND/OR INDUCE SEQUENCE CONVERGENCE,APL QUOTE-QUAD,VOL.2,NO.1,P.8-9 AVRIL 1970

(28) D.BIXLER=MSU APL,MICHIGAN STATE UNIVERSITY,COMPUTER LABORATORY,NOTICE NO.356,7 FEVRIER 1972.

(29) L.BOLLIET=EXPERIENCES D'ENSEIGNEMENT AVEC APL,COLLOQUE APL 9-10 SEPTEMBRE 1971,PUBLICATION IRIA P.445-60.

(30) D.A.BONYUM=MARK-SENSE APL,APL QUOTE-QUAD,VOL.3,NO.1,P.18-19, 11 JUIN 1971.

(31) W.G.BOURICIUS,W.C.CARTER,K.A.DUKE,J.P.ROTH ET P.R.SCHNEIDER= INTERACTIVE DESIGN OF SELF-TESTING CIRCUITRY,PROCEEDINGS OF THE PURDUE SYMPOSIUM ON INFORMATION PROCESSING,AVRIL 1969 P.73-80.

(32) P.BRAFFORT=EPI-APL,FUNDAMENTAL INSIGHTS FROM ADVANCES IN NOTATIONAL SYSTEMATICS,SEAS ANNUAL MEETING,PISE,SEPTEMBRE 1971.

(33) J.L.BRAME ET C.V.RAMAMOORTHY=AN INTERACTIVE SIMULATOR GENERATING SYSTEM FOR SMALL COMPUTERS,SJCC 1971,P.425-449.

(34) L.M.BREED ET R.H.LATHWELL=THE IMPLEMENTATION OF APL/360 , DANS ''INTERACTIVE SYSTEMS FOR EXPERIMENTAL APPLIED MATHEMATICS'',ACADEMIC PRESS 1968,P.390-399,A.KLERER ET J.REINFELDS EDITORS,ET ACM SYMPOSIUM ON EXPERIMENTAL SYSTEMS FOR APPLIED MATHEMATICS 1967.

(35) L.M.BREED ET R.H.LATHWELL=APL/360,IBM CONTRIBUTED LIBRARY 360-D-03-007,1968.

(36) L.M.BREED=GENERALIZING APL SCALAR EXTENSION,APL QUOTE-QUAD, VOL.2 ,NO.6,MARS 1971,P.5-7.

(37) L.M.BREED=DESIGN OF THE APL PLUS FILE SUB SYSTEM,COLLOQUE APL 9-10 SEPTEMBRE 1971,PUBLICATION IRIA.

(38) J.A.BROADSTON=CHARTING SCHEDULE PERFORMANCE,PRODUCTION AND INVENTORY MANAGEMENT, 1ST QUARTER 1970,P.79-81.

(39) J.A.BROWN=USING THE ACKERMAN FUNCTION TO RATE PROGRAMMING LANGUAGES,APL QUOTE-QUAD,VOL.2,NO.1,P.4-5,AVRIL 1970.

(40) J.A.BROWN=A GENERALIZATION OF APL.PH.D. THESIS,DEPARTMENT OF SYSTEMS AND INFORMATION SCIENCE,SYRACUSE UNIVERSITY,SEPTEMBRE 1971.

(41) E.A.BUCHHEIT ET R.B.RODEN=APL ROUTINES FOR EVALUATING FUNCTIONS IN MATHEMATICAL PHYSIC ,DEPT OF APPLIED ANALYSIS AND COMPUTER SCIENCE,RESEARC ,2029,NOVEMBRE 1970.

(42) P.CALINGAERT=INTRODU MING LANGUAGE,SCIENCE RESEARCH ASSOCIATES, EDITION,OCTOBRE 1967.

(43) R.S.CARBERRY ET COLL.=A MING LANGUAGE /1130, IBM CONTRIBUTED LIBRARY 1110-03-3-001,1968.

(44) V.CHAPTAL=COMPTE RENDU DU COLLOQUE APL,BULLETIN DE L'IRIA, NO.10,P.20-25,JANVIER 1972.

(45) V.CHAPTAL ET AL.=STRUCTURES ET SYSTEMES DE PROGRAMMATION, BULLETIN DE L'IRIA,NO.10,P.4-9,JANVIER 1972.

(46) S.CHARMONMAN,S.CABAY ET M.L.LOUIF-BYNE=USE OF APL/360 IN NUMERICAL ANALYSIS,DEPT OF COMPUTING SCIENCE,PUBLICATION NO.11,UNIVERSITY OF ALBERTA,EDMONTON,CANADA,DECEMBRE 1967.

(47) S.CHARMONMAN=A COMPARISON OF THE STRUCTURES OF APL,FORTRAN, ALGOL AND PL/1,APL QUOTE-QUAD,VOL.2,NO.4,P.2-4,JANVIER 1970.

(48) S.CHARMONMAN=SIXTY-CHARACTER REPRESENTATION OF APL SYMBOLS, APL QUOTE-QUAD,VOL.2,NO.2,P.5-10,10 JUILLET 1970.

(49) S.CHARMONMAN=A GENERALIZATION OF APL ARRAY ORIENTED CONCEPT, APL QUOTE-QUAD,VOL.2,NO.3,P.13-17,23 SEPTEMBRE 1970.

(50) S.CHARMONMAN=A GENERALIZATION OF APL ARRAY ORIENTED CONCEPT, SIGPLAN NOTICES,VOL.5,NO. 11,1970.

(51) S.CHARMONMAN=APL/UMC AN EXPERIMENTAL TRANSLATOR FOR BATCH PROCESSING OF A SUBSET OF APL,DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF MISSOURI-COLUMBIA,1971.

(52) S.J.CLARK=APL/360 AND 1130 VERSIONS-A COMPARISON,MC DONNELL DOUGLAS-ASTRONAUTICS,MATHEMATICAL SCIENCES DEPARTMENT,HUNINGTON BEACH,CALIFORNIA,20 DECEMBRE 1969,A3-950-L240 TECHNICAL MEMO 69-15.

(53) J.F.CLEMENTI ET P.P.FLETCHER=MODIFICATIONS TO THE APL/1130 SYSTEM TO PROVIDE MORE CONVENIENT OPERATING ON A FORTRAN USER'S MACHINE,APL QUOTE-QUAD,VOL.3,NO.1,P.16-18,11 JUIN 1971,ET VOL.3,NO.2/3,P.40-42, 1 OCTOBRE 1971.

(54) I.J.COLE=SOME APPLICATIONS OF A PROGRAMMING LANGUAGE,GODDARD SPACE FLIGHT CENTER,GREENBELT,MARYLAND,N67-37397,AOUT 1967.

(55) M.CORREIA,D.COSSMAN,F.PUTZOLU ET T.S.NETHEN=MINIMIZING THE PROBLEM OF LOGIC TESTING BY THE INTERACTION OF A DESIGN GROUP WITH USER ORIENTED FACILITIES,SEVENTH DESIGN AUTOMATION WORKSHOP,JUIN 1970,P.100-107.

(56) J.D.COUGER=SCHOOLS,COLLEGES ATTEST TO APL GROWTH,COMPUTERWORLD,VOL.6,NO. 13,P.16,29 MARS 1972.

(57) C.J.CREVELING=EXPERIMENTAL USE OF A PROGRAMMING LANGUAGE (APL) AT THE GODDARD SPACE FLIGHT CENTER,GSFC REPORT NO.X560-68-420,NOVEMBRE 1968,GREENBELT,MARYLAND.

(58) N.DAIRIKE=LAWRENCE RADIATION LABORATORY APL IMPLEMENTATION ON CDC 6000-7600,APL QUOTE-QUAD,VOL.3,NO.1,P.10-11,JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY,20-21 AVRIL 1971

(59) T.P.DANIELL=TIME-SHARING APL FOR IBM/1130 SYSTEMS,APL QUOTE-QUAD,VOL.3,NO.1,P.10-11,11 JUIN 1971 ET APL USERS' CONFERENCE WORKSHOP 3,BERKELEY,20-21 AVRIL 1971.

(60) I.DAVIDSON=SUMMARY OF CONFERENCE ON APRIL 6TH 1971 ON THE APPLICATION OF APL IN BELL NORTHERN RESEARCH AND BELL CANADA.

(61) W.H.E.DAY=COMPILER ASSIGNMENT OF DATA ITEMS TO REGISTERS, IBM SYSTEMS JOURNAL,NO.4 P.281-317,1970.

(62) M.DAYTON=A PLOTTER OF APL,APL QUOTE-QUAD,VOL.3,NO.1,11 JUIN 1971,P.13 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY 20-21 AVRIL 1971.

(63) G.DEMARS=SYSTEMES DE FICHIERS EN APL,RAPPORT INTERNE THOMSON-CSF,CCTI NO.2567,17 SEPTEMBRE 1971.

(64) G.DEMARS,J.C.RAULT ET G.RUGGIU=COMPTE RENDU DU COLLOQUE APL ORGANISE PAR L'IRIA LES 9 ET 10 SEPTEMBRE 1971,RAPPORT INTERNE THOMSON-CSF,LCR-DR5 NO.1607,23 SEPTEMBRE 1971.

(65) F.DESTOMBES=LE SYSTEME APL/360,COLLOQUE SUR LA TELEINFORMATIQUE 1969,TOME 1,P.414-421,EDITIONS CHIRON.

(66) G.DE VAHL DAVIS ET W.N.HOLMES=THE USE OF APL IN ENGINEERING EDUCATION,COLLOQUE APL,9-10 SEPTEMBRE 1971,PUBLICATION IRIA P.279-307.

(67) W.DE VRIES=WATCH YOUR(COMPUTER)LANGUAGE,THE ACTUARY,MARS 1971

(68) G.DHATT ET L.ROBICHAUD=FINITE ELEMENTS,FLOW -GRAPHS AND APL, COLLOQUE APL,9-10 SEPTEMBRE 1971,PUBLICATION IRIA P.37-69.

(69) K.A.DUKE,H.D.SCHNURMANN ET T.I.WILSON=SYSTEM VALIDATION BY THREE-LEVEL MODELING SYNTHESIS,IBM JOURNAL OF RESEARCH AND DEVELOPMENT,VOL.15,NO.2,P.166-74,MARS 1971.

(70) E.M.EDWARDS ET W.R.TINGA=AN APL COMPLEX ARITHMETIC PACKAGE, TECHNICAL APPLICATIONS PAPERS,NEREM 1970,P.106-112.

(71) E.M.EDWARDS=APL,A NATURAL LANGUAGE FOR ENGINEERING EDUCATION PART II,IEEE TRANSACTIONS ON EDUCATION,VOL.E-14,NO.4,P.179-80,NOVEMBRE 1971.

(72) D.W.EMBLEY=APL GRAPHICS,M.S. THESIS,DEPARTEMENT OF COMPUTER SCIENCE,UNIVERSITY OF UTAH, 1971.

(73) A.D.FALKOFF=ALGORITHMS FOR PARALLEL SEARCH MEMORIES,JOURNAL OF THE ACM,OCTOBRE 1962,P.488-511.

(74) A.D.FALKOFF,K.E.IVERSON ET E.H.SUSSENGUTH=A FORMAL DESCRIPTION OF SYSTEM/360,IBM SYSTEMS JOURNAL,VOL.3,NO.3,P.193-262, 1964.

(75) A.D.FALKOFF,K.E.IVERSON ET E.H.SUSSENGUTH=ERRATA FOR A FORMAL DESCRIPTION OF SYSTEM/360,IBM SYSTEMS JOURNAL,VOL.4, NO.1,P.84,1965.

(76) A.D.FALKOFF=FORMAL DESCRIPTION OF PROCESSES-THE FIRST STEP IN DESIGN AUTOMATION.PROCEEDINGS OF THE SHARE DESIGN AUTOMATION WORKSHOP,JUIN 1965.

(77) A.D.FALKOFF ET K.E.IVERSON=THE APL/360 TERMINAL SYSTEM P.22-37,DANS INTERACTIVE SYSTEMS FOR EXPERIMENTAL APPLIED MATHEMATICS,ACADEMIC PRESS,1968,M.KLERER ET J.REINFELDS EDITORS,ET ACM SYMPOSIUM ON EXPERIMENTAL SYSTEMS FOR APPLIED MATHEMATICS 1967,VOIR AUSSI IBM RESEARCH CENTER YORKTOWN, N.Y.,RC-1922,16 OCTOBRE 1967.

(78) A.D.FALKOFF ET K.E.IVERSON=THE APL TERMINAL SYSTEM =INSTRUCTIONS FOR OPERATIONS,IBM T.J.WATSON RESEARCH CENTER,YORKTOWN HEIGHTS,N.Y.10598,MARS 1967,REVU EN 1968.

(79) A.D.FALKOFF ET K.E.IVERSON=APL/360 USER'S MANUAL,IBM CORP., T.J.WATSON RESEARCH CENTER,YORKTOWN HEIGHTS,N.Y.,10598, FORM GH.20-0683.

(80) A.D.FALKOFF ET K.E.IVERSON=APL/360 MANUEL D'UTILISATION 1968 TRADUCTION FRANCAISE PAR Y.G.RAYNAUD ET G.P.SIMIAN,UNIVERSITE PAUL SABATIER,TOULOUSE,1970.

(81) A.D.FALKOF[illegible] [illegible]ITERIA FOR A SYSTEM DESIGN LANGUAGE,REPORT ON NATO SC[illegible] [illegible]MITTEE CONFERENCE ON SOFTWARE ENGINEERING TECHN[illegible]

(82) A.D.FA[illegible] K.E.IVERSON=APL/360-OS AND APL/360 DOS USER'S MANUAL,I[illegible]M FORM SH20-0906-0,FIRST EDITION,DECEMBRE 1970.

(83) A FALKOFF ET K.E.IVERSON=THE USE OF COMPUTERS IN TEACHING MATHEMATICS,IBM PHILADELPHIA SCIENTIFIC CENTER REPORT 320-2986,AVRIL 1970.

(84) A.D.FALKOFF=A SURVEY OF EXPERIMENTAL APL FILE AND I/O SYSTEMS IN IBM,COLLOQUE APL,9-10 SEPTEMBRE 1971,P[illegible]LICATION IRIA,P.365-74.

(85) P.FALSTER=APL IS A TOOL FOR THE FORMULATION OF PROBLEMS, DATABEHANDLING,NO.9,P.28-34,1970.

(86) J.FLETCHER=AN 8-BIT ASCII CODE,APL.QUOTE-QUAD,VOL.3,NO.1 P.13,11 JUIN 1971,ET APL USERS CONFERENCE WORKSHOP 3, BERKELEY,20-21 AVRIL 1971.

(87) P.H.FORTIN,D.SAMSON,P.LAVERDIERE ET L.P.A.ROBICHAUD=UTILISATION D'APL DANS LE CADRE DU PROJET DES STATUTS DU QUEBEC, COLLOQUE APL,9-10 SEPTEMBRE 1970,PUBLICATION IRIA P.115-137.

(88) G.H.FOSTER=APL A PERSPICUOUS LANGUAGE,COMPUTERS AND AUTOMATION,VOL.18,NO.12.P.24-26,28,NOVEMBRE 1969.

(89) G.H.FOSTER=USING APL TO INVESTIGATE SEQUENTIAL MACHINES, TECHNICAL APPLICATIONS PAPERS,NEREM 1970,P.121-7.

(90) G.H.FOSTER=APL,A NATURAL LANGUAGE FOR ENGINEERING PT I,IEEE TRANSACTIONS ON EDUCATION,VOL.E-14,NO.4,P.174-179,NOVEMBRE 1971.

(91) T.D.FRIEDMAN ET S.C.YANG=METHODS USED IN AN AUTOMATIC LOGIC DESIGN GENERATOR-(ALERT)IEEE.TRANS. ON COMPUTERS, VOL.C-18,NO.7,P.593-614,JUILLET 1969.

(92) D.C.GAZIS=A COMPUTER MODEL FOR THE FINANCIAL ANALYSIS OF

(93) URBAN PROJECTS,IBM RESEARCH CENTER,YORKTOWN HEIGHTS,N.Y., 13 AVRIL 1970,RC-2850.

(94) L.I.GILMAN ET A.J.ROSE=NOTES FOR THE VIDEOTAPE COURSE,IBM RESEARCH DIVISION,YORKTOWN HEIGHTS,N.Y..

(95) L.C.GILMAN ET A.J.ROSE=APL/360,AN INTERACTIVE APPROACH,IBM CORP.1969 ET J.WILEY 1970.

(96) N.GLICK ET R.SCHRADER=APL ON THE HONEYWELL 635.APL QUOTE-QUAD,VOL.3,NO.1,P.11,11 JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY 20-21 AVRIL 1971,APL QUOTE-QUAD,VOL.3, NO.2/3,P.20-30,1 OCT 1971.

(97) P.E.GRAY ET C.L.SEARLE=ELECTRONIC PRINCIPLES,PHYSICS,MODELS, AND CIRCUITS,J.WILEY 1969.

(98) L.GREENBERG=APL/6500,AT MICHIGAN STATE UNIVERSITY,APL QUOTE-QUAD,VOL.3,NO.1,P.20-21,11 JUIN 1971.

(99) P.GROSS,A.CROPLEY,B.HERB ET R.PALMER=APL AND REMOTE TERMINAL USAGE FOR COMPUTER ASSISTED INSTRUCTION,MANCHESTER DATA FAIR 1969.

(100) H.R.HAEGI=EULER A CAI-SYSTEM BASED ON APL,UNIVERSITY OF ZURICH,1971.

(101) P.E.HAGGERTY=AN APL SYMBOL SET FOR MODEL 35 TELETYPES,APL QUOTE-QUAD,VOL.2,NO.3,P.6-8,23 SEPTEMBRE 1970.

(102) A.HASSITT,J.W.LAGESHULTE ET L.E.LYON=A MICROPROGRAMMED IMPLEMENTATIO[illegible] AN APL MACHINE,APL QUOTE-QUAD,VOL.3,NO.1, P.[illegible]70 ET APL USERS CONFERENCE WORKSHOP 3, B[illegible]IL 1971.

(103) A.HASSITT,J.W.LAGESHULTE ET L.E.LYON=IMPLEMENTATION OF A HIGH LEVEL LANGUAGE MACHINE,PREPRINTS,ACM 4TH ANNUAL WORK-SHOP ON MICROPROGRAMMING,13-14 SEPTEMBRE 1971.

(104) A.HASSITT,J.W.LAGESHULTE ET L.E.LYON=A MICROPROGRAMMED APL MACHINE,COLLOQUE APL,PUB.IRIA,P.375-82,9-10 SEPT. 1971.

(105) A.HASSITT=MICROPROGRAMMING AND HIGH LEVEL LANGUAGES,INTER-NATIONAL IEEE COMPUTER CONFERENCE,P.91-92,SEPT.1971.

(106) A.HASSITT ET L.E.LYON=EFFICIENT EVALUATION OF ARRAY SUBSCRIPTS OF ARRAYS,IBM JOURNAL OF RESEARCH AND DEVELOPMENT VOL.16,NO.1,JANVIER 1972,P.45-57.

(107) W.S.HATCHER ET P.E.RETHIER=UNE APPLICATION DU LANGAGE APL AU PROBLEME DE DEMONSTRATION DE THEOREMES PAR ORDINATEUR,COLLO-QUE APL,PARIS 9-10 SEPTEMBRE 1971,PUBLICATION IRIA,P.443-443

(108) R.M.HEIBERGER=APL FUNCTIONS FOR DATA ANALYSIS AND STATISTICS RESEARCH REPORT CP-5,DEPT OF STATISTICS HARVARD UNIVERSITY, 31 MARS 1971.

(109) H.HELLERMAN=EXPERIMENTAL PERSONALIZED ARRAY TRANSLATOR SYSTEM,COMMUNICATIONS OF THE ACM,VOL.7,NO.7,PP.433-438, JUILLET 1964.

(110) H.HELLERMAN=DIGITAL COMPUTING SYSTEM PRINCIPLES,MC GRAW HILL 1967.

(111) J.C.HENSON ET W.F.MANRY=APL-AN INTRO,ATLANTA PUBLIC SCHOOLS ATLANTA,GEORGIA,2 NDE EDITION,AVRIL 1971.
VOIR APL QUOTE-QUAD,VOL.1,NO.3,P.3,OCTOBRE 1969.

(112) J.C.HERZ ET H.C.NGUYEN=APPLICATION DU LANGAGE APL A UN PROBLEME DE RECHERCHE OPERATIONNELLE,COLLOQUE APL,PUBL.IRIA P.141-56,9-10 SEPT.1971.

(113) J.A.HIGGINS=PROCEEDINGS OF THE APL USERS CONFERENCE AT S.U.N.Y.,BINGHAMPTON,JUILLET 1969.

(114) G.HORNE ET R.PIPER=A DESIGN FOR A 32-BIT COMPUTER USING APL STUDENTS AT POMONA COLLEGE,CLAREMONT,CALIFORNIA,MAI 1969.

(115) S.HUNKA=APL-A COMPUTING LANGUAGE DESIGNED FOR THE USER,THE BRITISH JOURNAL OF MATHEMATICAL AND STATISTICAL PSYCHOLOGY, VOL.20,PART.2,P.249-60,NOVEMBRE 1967.

(116) S.HUNKA=USE OF APL COMPUTER TERMINALS IN THE EDMONTON PUBLIC SCHOOLS.DIVISION OF EDUCATIONAL RESEARCH ,UNIVERSITY OF ALBERTA,EDMONTON,CANADA,1 AVRIL 1970-30 JUIN 1970 ET MARS 1971.

(117) D.HUTCHINSON=A NEW UNIFORM PSEUDORANDOM NUMBER GENERATOR, COMMUNICATIONS OF THE ACM ,VOL.9,NO.6,P.432-33,JUIN 1966.

(118) K.E.IVERSON=THE DESCRIPTION OF FINITE SEQUENTIAL PROCESSES INFORMATION THEORY,4TH LONDON SYMPOSIUM,C.CHERRY ED., BUTTERWORTHS 1961.

(119) K.E.IVERSON=A PROGRAMMING LANGUAGE,SJCC 1962,P.245-251, MAI 1962.

(120) K.E.IVERSON=A PROGRAMMING LANGUAGE,J.WILEY 1962.

(121) K.E.IVERSON=A COMMON LANGUAGE FOR HARDWARE,SOFTWARE AND APPLICATIONS,EASTERN JOINT COMPUTER CONFERENCE,P.121-9 (RC 749)DECEMBRE 1962.

(122) K.E.IVERSON=PROGRAMMING NOTATION IN SYSTEMS DESIGN,IBM SYSTEMS JOURNAL,VOL.2,NO.2,P.117-128,JUIN 1963.

(123) K.E.IVERSON=FORMALISM IN PROGRAMMING LANGUAGE,IBM CORP., T.J.WATSON RESEARCH CENTER,RC 992,2 JUILLET 1963.

(124) K.E.IVERSON=FORMALISM IN PROGRAMMING LANGUAGES,COMMUNICATIONS OF THE ACM,VOL.7,NO.2,P.80-88,FEVRIER 1964.

(125) K.E.IVERSON=RECENT APPLICATION OF A UNIVERSAL LANGUAGE,IFIP CONGRESS,NEW-YORK,24 MAI 1965 ET IBM RESEARCH RC 511,T.J. WATSON RESEARCH CENTER,YORKTOWN HEIGHTS,N.Y..

(126) K.E.IVERSON=ELEMENTARY FUNCTIONS,AN ALGORITHMIC TREATMENT, SCIENCE RESEARCH ASSOCIATES,CHICAGO,1966.

(127) K.E.IVERSON=THE ROLE OF COMPUTER IN TEACHING,QUEEN'S PAPERS ON PURE AND APPLIED MATHEMATICS,NO.13,1968,KIGNSTON,ONTARIO CANADA.

(128) K.E.IVERSON=THE USE OF APL IN TEACHING,IBM CORPORATION, FORM NO.320-0996-0,1969.

(129) K.E.IVERSON=THE USE OF APL IN STATISTICS,STATISTICAL COMPUTATION-PROCEEDINGS OF THE CONFERENCE AT THE UNIVERSITY OF WISCONSIN,AVRIL 1969,ACADEMIC PRESS,1969,R.C.MILTON ET J.A.NELDER EDS.,P.285-294.

(130) K.IVERSON ET A.D.FALKOFF=AN INTRODUCTION TO APL,NEREM 1970, P.97-98 (40 TITRES).

(131) K.E.IVERSON=THE STORY OF APL,COMPUTING REPORT,VOL.6,NO.2,P. 14-18,1970.

(132) K.E.IVERSON=ELEMENTARY ALGEBRA,IBM PHILADELPHIA SCIENTIFIC CENTER,TECHNICAL REPORT NO.320-3001,JUIN 1971.

(133) K.E.IVERSON=ALGEBRA AS A LANGUAGE,COLLOQUE APL,PARIS 9-10 SEPTEMBRE 1971,IRIA,P.5-16.

(134) K.E.IVERSON=APL IN EXPOSITION,TECHNICAL REPORT NO. 320-3010, IBM PHILADELPHIA SCIENTIFIC CENTER,1971.

(135) R.W.JAMIESON=ACT,AN ACTUARIAL PROGRAMMING LANGUAGE,SUN LIFE MONTREAL,QUEBEC,CANADA,1970.

(136) M.A.JENKINS=THE SOLUTION OF LINEAR SYSTEMS OF EQUATIONS AND LINEAR LEAST SQUARES PROBLEMS IN APL,IBM PHILADELPHIA SCIENTIFIC CENTER,TECHNICAL REPORT NO.320-2989,JUIN 1970.

(137) D.C.JESSUP=POWER-DELAY PRODUCT EVALUATION FOR COMBINATIONAL LOGIC CIRCUITS,IBM RESEARCH CENTER,YORKTOWN HEIGHTS,N.Y., 20 JUIN 1969,RC-2513.

(138) L.R.JOHNSON=SYSTEM STRUCTURE IN DATA,PROGRAMS AND COMPUTERS, PRENTICE HALL 1970.

(139) A.L.JONES=THE USE OF APL/360 IN MECHANICAL ANALYSIS, PROCEEDINGS OF THE 1970 IEEE INTERNATIONAL COMPUTER GROUP CONFERENCE,NEW-YORK,JUNE 1970,P.195-204.

(140) H.KATZAN=A PROSE GLOSSARY OF APL,COMPUTERS AND AUTOMATION, P.39-42,AOUT 1970.

(141) H.KATZAN=APL PROGRAMMING AND COMPUTER TECHNIQUES, VAN NOSTRAND 1970.

(142) H.KATZAN=APL USER'S GUIDE,VAN NOSTRAND,1971. (VOIR CRITIQUES IEEE ON C.,OCT.71,P.1222-3).

(143) H.KATZAN=REPRESENTATION AND MANIPULATION OF DATA STRUCTURES IN APL,PROCEEDINGS OF A SYMPOSIUM ON DATA STRUCTURES IN PROGRAMMING LANGUAGES,UNIVERSITY OF FLORIDA,GAINESVILLE, 25-27 FEVRIER 1971,J.T.TOU ET P.WEGNER EDS.,PUBLICATION ACM.

(144) G.KILDALL=EXPERIMENTS IN LARGE SCALE COMPUTER DIRECT ACCESS STORAGE MANIPULATION,TECH. REPT.NO.69-01-1,COMPUTER SCIENCE GROUP,UNIVERSITY OF WASHINGTON,SEATTLE WASHINGTON, JANVIER 1969.

(145) G.KILDALL=APL/B 5500 THE-LANGUAGE AND ITS IMPLEMENTATION. TECH.REPT. NO.70-09-04,COMPUTER SCIENCE GROUP,UNIVERSITY OF WASHINGTON,SEATTLE,WASHINGTON,SEPTEMBRE 1970.

(146) G.KILDALL=PRELIMINARY APL/B 5500 MANUAL,UNIVERSITY OF WASHINGTON,COMPUTER CENTER,SEATTLE,WASHINGTON-1970.

(147) G.KILDALL,L.SMITH,S.SWEDINE ET M.ZOZEL=UNIVERSITY OF WASHINGTON APL/B5500 MANUAL,COMPUTER SCIENCE GROUP,UNIVERSITY OF WASHINGTON,SEATTLE,TECHNICAL REPORT NO.71-1-10, JANVIER 1971.

(148) H.G.KOLSKY=PROBLEM FORMULATION USING APL,IBM SYSTEMS JOURNAL,VOL.8,NO.3,P.204-17,1969(G.231-0018).

(149) K.L.KONNERTH=USE OF A TERMINAL SYSTEM FOR DATA ACQUISITION, IBM JOURNAL OF RES.AND DEV.,VOL.13,NO.1,P.132-138,JANVIER 1969.

(150) K.L.KONNERTH ET M.L.PHILLIPS=APL/1130 WITH GRAPHIC AND OTHER I/O CAPABILITIES,IBM YORKTOWN HEIGHTS,20 JUILLET 1970, RC 2964.

(151) K.KORN=APL USERS CONFERENCE AT SUNY,DATAMATION,NOV.1969.

(152) R.J.KORSAN=A PROPOSED APL EXTENSION,APL QUOTE-QUAD,VOL.3, NO.2/3,P.21-23,1 OCT. 1971.

(153) S.E.KRUEGER ET T.P.MC MURCHIE=A PROGRAMMING LANGUAGE, SCIENCE RESEARCH ASSOCIATES,CHICAGO,1968.

(154) S.E.KRUEGER ET T.D.MC MURCHIE=APL/1500 USER'S GUIDE,SCIENCE RESEARCH ASSOCIATES,CHICAGO,1968.

(155) G.LACOURLY ET L.LEBART=ANALYSE MULTIDIMENTIONNELLE INTERACTIVE D'UN ENSEMBLE DE DONNEES,COLLOQUE APL,9-10 SEPTEMBRE 1971.

(156) R.H.LATHWELL=THE IMPLEMENTATION OF APL/360,VIEW GRAPHS, INTERNATIONAL SUMMER SCHOOL ON NEW TRENDS IN COMPUTER PROGRAMMING,20 AOUT 1968.

(157) R.H.LATHWELL=APL/360 OPERATIONS MANUAL,IBM CORP.,1968.

(158) R.H.LATHWELL=APL/360 SYSTEM GENERATION AND LIBRARY MAINTENANCE,IBM CORP.,1968,GH20-0683.

(159) R.H.LATHWELL ET J.E.MEZEI=A FORMAL DESCRIPTION OF APL,PUB. IRIA,P.181-215,COLLOQUE APL,9-10 SEPT.1971.

(160) B.A.LAWS=A PARALLEL BCH DECODER,ONR TECH.REP.,CONTRACT N,00014,67-C.0477,15 JUIN 1970.

(161) Y.LE BORGNE=APL/360 AU CENTRE D'ETUDES ET RECHERCHES D'IBM FRANCE,COLLOQUE APL,PUB. IRIA,P.239-50,9-10 SEPT.1971.

(162) Y.LE BORGNE=APL LANGAGE DE PROGRAMMATION DES MANAGERS,IBM INFORMATIQUE,1971.

(163) Y.LE BORGNE ET V.RISO=LE LANGAGE APL/360,UN OUTIL POUR L'INGENIEUR,L'ONDE ELECTRIQUE,VOL.51,FASC.11,P.899-904, DECEMBRE 1971.

(164) J.H.LEE=ADVANCED DECISION-MAKING FOR PRIVATE REAL ESTATE AND CONSTRUCTION MANAGEMENT-AN APL PROGRAM,PROCEEDINGS OF THE SOUTHWESTERN IEEE CONFERENCE,AVRIL 1970,P.272-76.

(165) H.A.LEKAN=INDEX TO COMPUTER ASSISTED INSTRUCTION,3RD EDITION HARCOURT BRACE JOVANOVIC,1971.

(166) A.R.LE PAGE=APL-A NATURAL LANGUAGE FOR ENGINEERING EDUCATION PT.3,IEEE TRANS. ON EDUCATION,VOL.E-14,NO.4,P.180-83, NOV. 1971.

(167) LE PENVEN,Y.RAYNAUD,G.SIMIAN=APL/CII 10070,RAPPORT DU MARCHE CRI 70.007,JUIN 1971.

(168) G.LE PENVEN,Y.RAYNAUD,G.SIMIAN ET H.MARTIN=APL/CII 10070, RAPPORT DU MARCHE CRI 70.007,DECEMBRE 1971.

(169) R.LIKNAITZKY=APL FUNCTIONS FOR USE IN JUNIOR HIGH SCHOOL MATHEMATICS,REPORT CAI 3-69,NOVEMBRE 1969,DIVISION OF EDUCATIONAL RESEARCH,FACULTY OF EDUCATION,THE UNIVERSITY OF ALBERTA,EDMONTON,ALBERTA,CANADA.

(170) Y.LIU=REVERSE OPERATOR IN APL,COMPUTING CENTER NEWS,VOL.4, NO.4,P.9-10,SYRACUSE UNIVERSITY,1 MARS 1971.

(171) R.A.LORIE=APL AS A LANGUAGE FOR HANDLING A RELATIONAL DATA-BASE,IBM CORP.,CAMBRIDGE SCIENTIFIC CENTER,G320-2067, MARS 1971.

(172) R.A.LORIE ET A.J.SYMONDS=USE OF A RELATIONAL ACCESS METHOD UNDER APL,IBM CAMBRIDGE SCIENTIFIC CENTER,G320-2071, MAI 1971,ET SYMPOSIUM ON DATA BASE SYSTEMS,COURANT INSTITUTE OF MATHEMATICAL SCIENCES,MAI 1971.

(173) R.A.LORIE ET A.J.SYMONDS=INTERACTIVE PROBLEM SOLVING USING A RELATIONAL DATA-BASE IN APL,1971 INTERNATIONAL IEEE COMPUTER CONF.,P.191-92.

(174) G.LOTTO=ON-LINE ANALYSIS OF SURVEY DATA,IBM ASSD,MOHANSIC, DEPARTMENT 983.

(175) T.LUTZ=APL-PROFILE OF A DIALOGUE LANGUAGE,COMPUTER PRAXIS, VOL.4,NO.4,P.66-73,AVRIL 1971.

(176) T.MAC AULEY=CAL/APL COMPUTER ASSISTED LEARNING,A PROGRAMMING LANGUAGE AUTHOR'S MANUAL,INFORMATION SERVICES AND COMPUTER FACILITY,ORANGE COAST JUNIOR COLLEGE DISTRICT,COSTA MESA, CALIFORNIA,MAI 1969.

(177) H.P.MACON=A SURVEY OF APL COMPATIBLE TERMINALS,APL QUOTE-QUAD,VOL.3,NO.2/3,P.12-20,1 OCT.1971.

(178) G.MARTIN=UN LANGAGE DE MANIPULATION FORMELLE,PUB.IRIA, P.405-431,COLLOQUE APL,9-10 SEPTEMBRE 1971.

(179) H.MARTIN=SIMULATION EN APL DES COMMANDES D'UN SYSTEME DE GESTION DE FICHIERS POUR LE SYSTEME APL,CENTRE D'INFORMATIQUE DE TOULOUSE,JUIN 1971.

(180) P.MAURICE=DESCRIPTION DU SYSTEME 360 A L'AIDE DE LA NOTATION D'IVERSON,DIPLOME D'INGENIEUR ENSEIHT,TOULOUSE 1968.

(181) P.MAURICE ET P.C.SCHOLL=UN INTERPRETEUR DU LANGAGE APL POUR LE CII 90-80,BULLETIN DE L'IRIA,P.99- [illegible]

(182) P.MAURICE,Y.RAYNAUD ET G.SIMIAN=REALISATION D'APL A L'UNIVERSITE DE TOULOUSE,COLLOQUE APL,9-10 SEPTEMBRE 1971.

(183) G.K.MC AULIFFE=APL FAST FOURIER PROGRAM,IBM RESEARCH CENTER YOKTOWN HEIGHTS,N.Y.,MARS 1970,RC-2832.

(184) D.MC.CRACKEN=WHITHER APL-DATAMATION,VOL.16,NO.11,P.53-55, 15 SEPTEMBRE 1970.

(185) A.MC EWAN ET D.WATSON=APL/360 RECURSED,PART.1.APL QUOTE-QUAD VOL.2,NO.2,P.11-16,10 JUILLET 1970.

(186) T.D.MC MURCHIE=APPLAUSE FOR APL,COMPUTERS AND AUTOMATION, VOL.19,NO.3,P.[illegible],MARS 1970.

(187) T.D.MC MURCHIE ET S.E.KRUEGER ET H.T.LIPPERT=A PROGRAMMING LANGUAGE/1500,AD.716-733,30 NOVEMBRE 1970.

(188) T.D.MC MURCHIE=A LIMITED CHARACTER APL SYMBOLISM,SIGPLAN NOTICES,VOL.6,NO.1,1971.

(189) T.D.MC MURCHIE ET D.B.THOMAS=APL/1500 FILE ACCESS SUBROUTINE PACKAGE,AD.717-928,1ER FEVRIER 1971.

(190) T.D.MC MURCHIE ET D.B.THOMAS=MANUAL OF APL/1500 FUNCTIONS-SYSTEM FUNCTIONS AD-717-737,1 FEVRIER 1971.

(191) H.MELMS=APL FOR GOVERNMENT TECHNOLOGY PROBLEMS,IBM NACHR., VOL.21,NO.205,P.646-54,FEVRIER 1971.

(192) M.S.MONTALBANO=TIME-SHARED CRITICAL PATH CALCULATIONS, 320-3219,AOUT 1967.

(193) M.MONTALBANO=HIGH-SPEED CALCULATION OF THE CRITICAL PATHS OF LARGE NETWORKS,IBM SYSTEMS JOURNAL,VOL.6,NO.3,P.163-191, 1967.

(194) M.S.MONTALBANO=CONVERSATIONAL LINEAR PROGRAMMING-A USER'S MANUAL FOR LPAPL,COMPUTERS IN MANAGEMENT EDUCATION,REPORT NO.1,IBM PALO ALTO SCIENTIFIC CENTER,320-3272,MARS 1970.

(195) R.MORE,Y.RAYNAUD ET G.SIMIAN=UN INTERPRETEUR EN MODE CONVERSATIONNEL POUR LA NOTATION D'IVERSON,APPLICATION A LA DESCRIPTION FORMELLE DE SYSTEMES,GRENOBLE,OCTOBRE 1967.

(196) R.MORE,Y.G.RAYNAUD ET G.P.SIMIAM=UN LANGAGE CONVERSATIONNEL POUR L'AIDE A LA CONCEPTION ET A LA REALISATION DES SYSTEMES INFORMATIQUES,COLLOQUE SUR LA MICROELECTRONIQUE-TOULOUSE, MARS 1969.

(197) R.MORE,Y.G.RAYNAUD ET SIMIAN=UN LANGAGE DE PROGRAMMATION CONVERSATIONNEL,COLLOQUE INTERNATIONAL SUR LA TELE-INFORMATIQUE,P.158,TOME II,EDITIONS CHIRON,1969.

(198) R.MORE=CONTRIBUTION A LA REALISATION D'UN INTERPRETEUR APL CONVERSATIONNEL,THESE UNIVERSITE DE TOULOUSE,JANVIER 1971.

(199) A.MUKHOPADHYAY ET G.SCHMITZ=MINIMIZATION OF EXCLUSIVE-OR AND LOGICAL EQUIVALENCE SWITCHING NETWORKS.IEEE TRANS. ON COMPUTERS,VOL.C-19,NO.2,P.132-40,FEVRIER 1970.

(200) H.J.MYERS ET M.Y.HSIAO=AN APL ALGORITHM FOR CALCULATING BOOLEAN DIFFERENCE.AUTOMATIC SUPPORT SYSTEMS SYMPOSIUM FOR ADVANCED MAINTAINABILITY,ST-LOUIS,MO,NOVEMBRE 1968, P.5 D-1 A 5 D-1[illegible].

(201) W.H.NIEHOFF=A HYPOTHETICAL 32-BIT PROCESSOR FOR SYSTEMS TRAINING-ITS APL/360 DESCRIPTION AND SIMULATION,IBM SYSTEMS DEVELOPMENT DIVISION,ENDICOTT,N.Y.,22 MAI 1970,TR 01.1316.

(202) J.L.OWENS=BULK I/O AND COMMUNICATIONS WITH LIVERMORE TIME SHARING SYSTEM.APL QUOTE-QUAD,VOL.3,NO.1,P.7-8,11 JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY 20-21 AVRIL 1971

(203) G.L.NOGUEZ ET D.M.PECCOUD=AN ARRAY PROCESSOR DESIGN FOR APL-LIKE DATA STRUCTURE,IFIP CONGRESS 1971,BOOKLET TA-4.

(204) P.D.PAGE=AN ON-LINE PROOF CHECKER OPERATING UNDER APL/360-APL QUOTE-QUAD,VOL.3,NO.1,P.4-5,11 JUIN 1971 ET APL USER'S CONFERENCE WORKSHOP 3,BERKELEY 20-21 AVRIL 1971,APL QUOTE-QUAD,VOL.3,NO.2/3,P.30-34,1 OCT.1971.

(205) S.PAKIN=APL/360 REFERENCE MANUAL,SCIENCE RESEARCH ASSOCIATES PALO ALTO,CALIFORNIA,2 NDE EDITION,1971.

(206) J.P.PAQUET=SIMPLIFICATION DES FONCTIONS BOOLEENNES A L'AIDE DES MATRICES A N DIMENSIONS-THESE UNIVERSITE LAVAL,QUEBEC, CANADA 1968.

(207) D.M.PECCOUD ET G.L.NOGUEZ=AN ARRAY PROCESSOR DESIGN FOR APL LIKE DATA STRUCTURES,IFIP CONGRESS 1971,NORTH HOLLAND.

(208) P.PENFIELD=MARTHA USER'S MANUAL,ELECTRODYNAMICS MEMO NO.6, 21 SEPT. 1970,MIT RESEARCH LABORATORY OF ELECTRONICS.

(209) P.PENFIELD=ADDENDUM TO MARTHA USER'S MANUAL,ELECTRODYNAMICS MEMO NO.12,13 NOV.1970,MIT RESEARCH LABORATORY OF ELECTRONICS.

(210) P.PENFIELD=GENERAL PURPOSE ELECTRIC-CIRCUIT ANALYZER IMBEDDED IN APL.ELECTRODYNAMICS MEMO NO.15,RESEARCH LAB.OF ELECTRONICS,MIT,26 FEVRIER 1971.

(211) P.PENFIELD=A SET OF APL PROGRAMS FOR USE IN NETWORK THEORY, APL QUOTE-QUAD,VOL.3,NO.1,P.4,11 JUIN 1971 ET APL USER'S CONFERENCE WORKSHOP 3,BERKELEY,20-21 AVRIL 1971.

(212) P.PENFIELD=GENERAL PURPOSE NETWORK ANALYSIS USING WIRING OPERATORS,IEEE CONFERENCE ON ELECTRICAL NETWORK THEORY,1971, P.116-117.

(213) P.PENFIELD=MARTHA USER'S MANUAL,THE MIT PRESS 1971.

(214) P.PENFIELD=DESCRIPTION OF ELECTRICAL NETWORKS USING WIRING OPERATORS,PROCEEDINGS OF THE IEEE,VOL.60,NO.1,P.49-53, JANVIER 1972.

(215) A.PERLIS=APL AS A CONVENTIONAL LANGUAGE-WHAT IS MISSING APL QUOTE-QUAD,VOL.3,NO.1,P.3-4,11 JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY 20-21 AVRIL 1971.

(216) A.J.PERLIS,R.D.FENNELL,F.J.POLLACK,W.R.PRICE ET M.F.RIZZO= CONVERSATIONAL PROGRAMMING-APL,AN IMPLEMENTATION IN BLISS, AD.729-941,JUIN 1971.

(217) J.PLOTKE=MINIPERT,A TERMINAL CONTROLLED CRITICAL PATH TECHNIQUE,IBM SMD,DEPARTMENT 847,DIVISION 32,HARRISON.

(218) [illegible]ERTOUZOS,I.LEE ET K[illegible]TH= ON IMPORTANT CURRE[illegible]UES CONCERNING COMPUTERS IN ELECTRICAL ENGINEER- EDUCATION,IEEE TRANSACTIONS ON EDUCATION,VOL.E-14,NO.4, P.169-74,NOVEMBRE 1971.

(219) W.PRAGER=AN INTRODUCTION TO APL,ALLYN AND BACON INC.,1970.

(220) T.H.PUCKETT=NOTES ON THE INSTALLATION OF APL/OS,NEW MEXICO STATE UNIVERSITY,LAS CRUCES,NEW MEXICO,REPORT(505) 646-3439

(221) S.M.RAUCHER=INTRODUCTION TO APL-VIDEOTAPES,IBM CORP.,1968.

(222) S.M.RAUCHER=APL AND ITS USE IN THE CLASSROOM,JOURNAL OF THE ASSOCIATION FOR EDUCATION DATA SYSTEMS,DECEMBRE 1968.

(223) J.C.RAULT=SYSTEMES APL,RAPPORT INTERNE THOMSON-CSF,LCR-DR5, NO.1548,MARS 1971.

(224) Y.RAYNAUD=APL,SON IMPLANTATION,SON UTILISATION POUR L'AIDE A LA CONCEPTION DES SYSTEMES DE TRAITEMENT DE L'INFORMATION, BULLETIN DE L'IRIA,MARS 1971,P.6-98.

(225) R.J.D.REEVES=APL,A POTENTIAL LIABILITY,DATAMATION, 15 SEPTEMBRE 1971,P.71-72.

(226) H.A.REICH=AN EXPERIMENTAL SYSTEM FOR TIME-SHARED,ON-LINE DATA ACQUISITION,IBM JL OF RES. AND DEV.,VOL.13,NO.1,P.114-118,1969.

(227) B.ROBINET=SUR UN LANGAGE CONVERSATIONNEL,PROGRES ET SCIENCE, NO.4,1970.

(228) B.ROBINET=LE LANGAGE APL,OU L'ART DE PROGRAMMER EN LIBERTE, 01-INFORMATIQUE,P.45-50,11 DECEMBRE 1970.

(229) B.ROBINET,B.ARLETTAZ,J.C.GIRARD ET J.MICHEL=LE LANGAGE D'IVERSON,RAPPORT DGRST NO.69.01-586,PARIS,JUIN 1971.

(230) B.ROBINET=SEMANTIQUE D'APL,COLLOQUE APL,PUB.IRIA,P.217-232, 9-10 SEPT.1971.

(231) B.ROBINET,J.C.GIRARD ET B.ARLETTAZ=UN COMPILATEUR INCRE- MENTIEL D'APL,COLLOQUE APL,PUB.IRIA,P.315-337,9-10 SEPT.1971

(232) B.ROBINET=LE LANGAGE APL,EDITIONS TECHNIP 1971.

(233) B.ROBINET=SEMANTIQUE DES TABLEAUX-APPLICATION AU LANGAGE APL,THESE DE 3EME CYCLE,UNIVERSITE DE PARIS VI,28 FEVRIER 1972.

(234) W.M.RODGERS=PART 4 -A PRELIMINARY SURVEY OF GRAPHICAL DISPLAY SYSTEMS,AD-716-593,JUIN 1970.

(235) A.J.ROSE=VIDEOTAPED APL COURSE,IBM CORP.,1967.

(236) A.J.ROSE=TEACHING THE APL/360 TERMINAL SYSTEM,IBM CORP.,RC 2184,28 AOUT 1968,T.J.WATSON RESEARCH CENTER,YORKTOWN HEIGHTS N.Y..

(237) A.J.ROSE=APL FOR USERS OF BASIC,SCIENTIFIC TIMESHARING CORP.,WASHINGTON,D.C..

(238) B.ROSENKRANDS=APL EXERCISES,IBM DENMARK.

(239) B.ROSENKRANDS=GRAPHICS BY APL,PUB.IRIA,P.91-113,COLLOQUE APL 9-10 SEPTEMBRE 1971.

(240) J.P.ROTH=DIAGNOSIS OF AUTOMATA FAILURES,A CALCULUS AND A METHOD,IBM JOURNAL OF RESEARCH AND DEVELOPMENT,VOL.10, P.278-91,JUILLET 1966.

(241) J.P.ROTH,W.G.BOURICIUS ET P.R.SCHNEIDER=PROGRAMMED ALGORITHMS TO COMPUTE TESTS TO DETECT AND DISTINGUISH BETWEEN FAILURES IN LOGIC CIRCUITS,IEEE TRANSACTIONS ON EC, VOL,EC-16,NO.5,P.567-80,OCT.1967.

(242) D.RUDBERG,D.BRUNSVOLD ET M.HITCH=APL/MSU-BTM,USER'S MANUAL, A SUPPLEMENT TO THE APL/360,REFERENCE MANUAL,OR APL/360 PRIMER,AUTOMNE 1970.

(243) D.RUDBERG=APL,A NATURAL LANGUAGE FOR ENGINEERING EDUCATION PT IV,IEEE TRANS. ON EDUCATION,VOL.E-14,NO.4,P.183-85, NOV. 1971.

(244) D.RUETER=ARRAY FOR APL-DATAMATION,15 NOV.1971,P.17.

(245) G.RUGGIU=SEMANTIQUE DES LANGAGES DE PROGRAMMATION ET INTERPRETATION GLOBALE DES EXPRESSIONS,C.R. ACAD. SC.,PARIS, TOME 273-SERIE A,P.1271-1274,20 DECEMBRE 1971 ,ET TOME 274 SERIE A,P.100-103,3 JANVIER 1972.

(246) G.RUGGIU=DESCRIPTION SEMANTIQUE DES FONCTIONS PRIMITIVES D'APL,REVUE TECHNIQUE THOMSON-CSF,MARS 1972.

(247) J.RYAN=APL/700,AN APL IMPLEMENTATION FOR THE BURROUGHS 6700 AND 7700-APL QUOTE-QUAD,VOL.3,NO.1,P.12,11 JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY 20-21 AVRIL 1971

(248) J.RYAN=GENERALIZED LISTS AND OTHER EXTENSIONS,APL QUOTE-QUAD VOL.3,NO.1,P.8-10 JUIN 1971.

(249) J.SAMMET=PROGRAMMING LANGUAGES,HISTORY AND FUNDAMENTALS, PRENTICE HALL,1969,P.247-53.

(250) D.SANT=THE M RX 1240 COMMUNICATION TERMINAL AND 1270 TRANSMISSION CONTROL UNIT,APL QUOTE-QUAD,VOL.3,NO.1,P.13,11 JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY, 20-21 AVRIL 1971.

(251) R.O.SCHMIDT=A COLLECTION OF WALSH ANALYSIS PROGRAMS,IEEE TRANSACTIONS ON ELECTROMAGNETIC COMPATIBILITY,VOL.EMC-13, NO.3,P.88-94,AOUT 1971.

(252) P.SCHOLL=PROBLEMES RELATIFS A L'ANALYSE SYNTAXIQUE DE LA NOTATION D'IVERSON,DIPLOME D'INGENIEUR ENSEIHT,TOULOUSE,1968

(253) P.SCHOLL ET Y.RAYNAUD=PROBLEMES RELATIFS A L'ANALYSE SYNTAXIQUE DE LA NOTATION D'IVERSON,CENTRE D'INFORMATIQUE DE TOULOUSE,ANNEE 1967-68.

(254) G.P.SCHREIBER ET R.POLIVKA=EXPERIENCES AND OBSERVATIONS WITH A SELF-TEACHING COURSE IN APL,COLLOQUE APL, 9-10 SEPTEMBRE 1971,PUB.IRIA,P.77-90.

(255) C.SEABERG=COMPUTER ASSISTED FORECASTING-HOW BUSINESS IS USING APL-CANADIAN DATASYSTEMS ,VOL.3,NO.1,P.30-31, JANVIER 1971.

(256) C.SEABERG=APL IN FINANCIAL FORECASTING IS BASE FOR EVENTUAL MIS,CANADIAN DATASYSTEMS,VOL.3,NO.2,P.50-53,FEVRIER 1971.

(257) C.L.SEARLE=TEACHING OF TRANSISTOR CIRCUIT DESIGN USING A DIGITAL COMPUTER,IEEE TRANSACTIONS ON EDUCATION,VOL.E12,NO.3 P.216-22,SEPTEMBRE 1969.

(258) C.L.SEARLE=APL,A NATURAL LANGUAGE FOR ENGINEERING EDUCATION PT IV IEEE TRANS. ON EDUCATION,VOL.E-14,NO.4,P.185,NOV.1971.

(259) C.L.SEARLE=TRANSISTOR AMPLIFIER DESIGN-A STUDY IN WHEN NOT TO USE THE COMPUTER,IEEE TRANSACTIONS ON EDUCATION,1972.

(260) E.SHARON=AN APL/360 INCOME TAX PROGRAM,THE DESCRIPTION OF DATA PROCESSING PROCEDURES,IBM CORP.,MAI 1968,320-3242.

(261) I.P.SHARP=A BRIEF HISTORY OF APL,CANADIAN DATASYSTEMS,P.44-47 ET 74,FEVRIER 1970.

(262) I.P.SHARP=THE FUTURE OF APL TO BENEFIT FROM A NEW FILE SYSTEM,CANADIAN DATASYSTEMS,P.44-45,85,MARS 1970.

(263) K.W.SMILLIE=SOME APL PROGRAMS FOR STATISTICAL CALCULATIONS, DEPT. OF COMPUTING SCIENCE,UNIVERSITY OF ALBERTA,EDMONTON, CANADA,PUBLICATION NO.6,1967.

(264) K.W.SMILLIE=STATPACK 1,AN APL STATISTICAL PACKAGE,PUB.NO.9, DEPT. OF COMPUTING SCIENCE,UNIVERSITY OF ALBERTA,EDMONTON, CANADA,JANVIER 1968.

(265) K.W.SMILLIE=AN APL ALGORITHM FOR THE CRITICAL PATH,QUATERLY BULLETIN OF THE COMPUTER SOCIETY OF CANADA,VOL.8,NO 2, P.6-13,PRINTEMPS 1968.

(266) K.W.SMILLIE=STATPACK 2,AN APL STATISTICAL PACKAGE,DEPT OF COMPUTING SCIENCE,UNIVERSITY OF ALBERTA,PUB.17,FEVRIER 1969, EDMONTON,ALBERTA,CANADA.

(267) K.W.SMILLIE=SOME APL ALGORITHMS FOR ORTHOGONAL FACTORIAL EXPERIMENTS,DEPT. OF COMPUTING SCIENCE.PUBLICATION NO.18, UNIVERSITY OF ALBERTA,EDMONTON,ALBERTA,CANADA,JUIN 1969.

(268) K.W.SMILLIE=THE APL LANGUAGE AND STATISTICAL COMPUTATIONS, COMPUTER BULLETIN,VOL.13,NO.8,P.896-897,AOUT 1969.

(269) K.W.SMILLIE=AN INTRODUCTION TO APL/360 WITH SOME STATISTICAL APPLICATIONS,DEPT OF COMPUTING SCIENCE,PUBLICATION NO.19, UNIVERSITY OF ALBERTA,EDMONTON,ALBERTA,CANADA,JANVIER 1970.

(270) K.W.SMILLIE=STATISTICAL PROGRAMS IN APL/360,COMPUTER BULLETIN,VOL.14,NO.5,P.151-152,MAI 1970.

(271) K.W.SMILLIE=APL/360 WITH STATISTICAL EXAMPLES(NON PUBLIE) 1971.

(272) K.W.SMILLIE=APL AND STATISTICS,PROGRAMS OR INSIGHT,COLLOQUE APL,PUBLICATION IRIA,P.17-35,PARIS,9-10 SEPT.1971.

(273) V.N.SPADAVECCHIA,P.C.BERRY ET G.BARTOLI=AN ABSTRACT MACHINE FOR THE INTRODUCTION TO COMPUTER SCIENCE,COLLOQUE APL,9-10 SEPT. 1971,PUBLICATION IRIA,P.273-78.
(VOIR BERRY RAPPORT PUBLIE,FEVRIER 1972).

(274) T.A.STANDISH=AN ESSAY ON APL,DEPARTMENT OF COMPUTER SCIENCE, CARNEGIE-MELLON UNIVERSITY,PITTSBURGH,MARS 1969.

(275) G.P.STICKELER=REAL-WORLD APL,DATAMATION,1 DEC. 1971,P.19.

(276) R.K.STOCKWELL ET K.E.VAN BEE=USE OF APL TO IMPLEMENT ALGORITHMS FOR SPARSE LINEAR SYSTEMS,NEREM 1970,P.113-119.

(277) E.A.STOHR=SIMULATION OF SOME APL OPERATORS,REPORT LR-16, FEVRIER 1971,CENTER FOR RESEARCH IN MANAGEMENT SCIENCE, UNIVERSITY OF CALIFORNIA,BERKELEY,CALIFORNIA.

(278) O.STUTZ=APL/360 A TIME-SHARING SERVICE WITH A MODERN PROBLEM LANGUAGE.APL/360 A FORM OF SUBSCRIBER OPERATION WITH A MODERN PROBLEM LANGUAGE,IBM NACHRICHTEN,
VOL.20,NO.199,P.79-83,FEVRIER 1970.
VOL.20,NO.200,P.164-9,AVRIL 1970.

(279) Y.SUNDBLAD=THE ACKERMAN FUNCTION,A THEORETICAL COMPUTATIONAL AND FORMULA MANIPULATIVE STUDY,BIT,VOL.11,NO.1,1971.

(280) A.J.SURKAN=SYMBOLIC POLYNOMIAL OPERATIONS WITH APL,IBM JOURNAL OF RESEARCH AND DEVELOPMENT,VOL.13,NO.2,P.209-211 MARS 1969.

(281) A.J.SURKAN=DISCRETE FAST FOURIER TRANSFORMATION MADE SIMPLE BY A SINGLE RELIABLE APL FUNCTION,IBM RESEARCH CENTER, YORKTOWN HEIGHTS,N.Y.,22 AOUT 1969,RC-2591.

(282) Y.TALLINEAU=QUELQUES REFLEXIONS SUR LE COLLOQUE APL A L'IRIA,INFORMATIQUE ET GESTION,NOV.71,P.80.

(283) Y.TALLINEAU=L'APL UN LANGAGE ADAPTE A LA GESTION, INFORMATIQUE ET GESTION,NO.25,P.79-82,FEVRIER 1971.

(284) A.TAYLOR=APL,A COMPLEX OR SIMPLE LANGUAGE,COMPUTERWORLD, 1 AVRIL 1970.

(285) W.G.THISTLE AND D.S.GALBRAITH=DIFFERENCES BETWEEN DREV APL AND IBM APL/360,REPORT DREV M.2118/71,DEFENSE RESEARCH ESTABLISHMENT,VALCARTIER,QUEBEC,CANADA.

(286) R.D.THORNTON=COMPUTER-FLAVORED CIRCUIT THEORY,IEEE TRANSACTIONS ON EDUCATION,VOL.E-12,NO.3,P.219-222,SEPTEMBRE 1969.

(287) K.J.THURBER ET J.W.MYRNA=SYSTEM DESIGN OF A CELLULAR APL COMPUTER IEEE TRANSACTIONS ON COMPUTERS,VOL.C-19,NO.4,P.291-303,AVRIL 1970.

(288) B.TUCKERMAN=A STUDY OF THE VIGENERE-VERNAM SINGLE AND MULTIPLE LOOP ENGINEERING SYSTEMS,IBM RESEARCH REPORT (RC 2879)MAI 1970.

(289) H.VAN HEDEL=AN APL BATCH PROCESSOR,COLLOQUE APL,PUB.IRIA, P.339-64,9-10 SEPT.1971.

(290) U.M.VON MAYDELL=AN INTRODUCTION TO PROBABILITY USING APL, DEPT.OF COMPUTING SCIENCE,PUBLICATION NO.21,UNIVERSITY OF ALBERTA,EDMONTON,ALBERTA,CANADA,JUIN 1970.

(291) P.N.WAHI=AIMS-APPLIED INFORMATION AND MANAGEMENT SIMULATION, A GENERAL BUSINESS SIMULATION IN APL,IBM CORP.,CAMBRIDGE SCIENTIFIC CENTER,G320-2066,AVRIL 1971.

(292) J.WILLIAMS=CONDITIONAL BRANCH APL COMPILER,APL QUOTE-QUAD, VOL.3,NO.1,P.5-6,11 JUIN 1971 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY,20-21 AVRIL 1971.

(293) L.J.WOODRUM=INTERNAL SORTING WITH MINIMAL COMPARING,IBM SYSTEMS JOURNAL,VOL.8,NO.3,P.189-203,1969.

(294) L.J.WOODRUM=A MODEL OF FLOATING BUFFERING,IBM SYSTEMS JOURNAL,VOL.9,NO.2,P.118-144,1970.

(295) R.ZAKS,D.STEINGART ET J.MOORE=A FIRMWARE APL TIMESHARING SYSTEM,SJCC 1971,P.179-90.

(296) R.ZAKS ET D.STEINGART=A LANGUAGE MACHINE,APL QUOTE-QUAD, VOL.3,NO.1,P.6 ET APL USERS CONFERENCE WORKSHOP 3,BERKELEY, 20-21 AVRIL,APL QUOTE-QUAD,VOL.3,NO.2/3,P.34-39,1 OCT.71.

(297) R.ZAKS=MICROPROGRAMMED APL,INTERNATIONAL IEEE COMPUTER CONFERENCE,P.192-4,SEPT. 1971.

(298) M.ZOZEL=UNIVERSITY OF WASHINGTON IMPLEMENTATION,APL QUOTE-QUAD,VOL.2,NO.4,P.6-7,NOVEMBRE 1970.

(299) J.A.HIGGINS=PROCEEDINGS OF THE APL USERS CONFERENCE AT SUNY BINGHAMPTON,JUILLET 1969.

(300) APL USERS CONFERENCE WORKSHOP 3,BERKELEY,20-21 AVRIL 1971, VOIR APL QUOTE-QUAD,VOL.3,NO.1,P.3-13,11 JUIN 1971.

ANONYMOUS

(301) KIDNEY MATCHED BY COMPUTER,NEWS BRIEFS,DATAMATION AVRIL 1970,P.215-16.

(302) STORY OF APL-AN INTERVIEW WITH DR K.IVERSON,COMPUTERWORLD, 1 AVRIL 1970.

(303) CREATING PLAIN TALK FOR COMPUTERS,A PASSION FOR PRECISION LEADS KEN IVERSON TO AN EASY-TO-USE LANGUAGE,IBM MAGAZINE, VOL.2,NO.3,P.11-12;16 FEVRIER 1970.

(304) COMPUTING NEWSLETTERS FOR INSTRUCTORS OF DATA PROCESSING , MARS 1971 ET 1972.

(305) COMPUTERWORLD=EDITORIAL,16 AVRIL 1969.

(306) LE CULTE DE L'APL=INFORMATIQUE ET GESTION,OCT. 1971,P.11-12.

(307) UN TERMINAL SPECIAL APL INFORMATIQUE ET GESTION,OCTOBRE 1971 P.24,(TERMINAL OLIVETTI TE.338,APL).

PERIODICALS

(308) APL NEWSLETTERS,I.P.SHARP ASSOCIATES,MONTHLY PUBLICATION IN CANADIAN DATASYSTEMS.

(309) APL NEWSLETTER=W.JURAN EDITOR,PROPRIETARY COMPUTER SYSTEMS, 1662 SOUTH SATICOY STREET,VAN NUYS,CALIFORNIA,91406.

(310) APL QUOTE-QUAD
EDITORS=A.T.MC.EWAN ET D.W.A.WATSON
LAKEHEAD UNIVERSITY,THUNDER BAY,ONTARIO
DITRIBUTOR=G.H.FOSTER,SYRACUSE UNIVERSITY,SYRACUSE, NEW-YORK,USA.

(311) SEAS APL WORKING COMMITTEE=

SECRETARY=NIELS GELLERT,NEUCC,TECHNICAL UNIVERSITY OF DENMARK,2800 LYNGBY,DENMARK.

CHAIRMAN=P.S.ABRAMS,CEGOS-INFORMATIQUE,SERVICE APL, 14 RUE ANATOLE FRANCE,92 PUTEAUX,FRANCE.

IBM BROCHURES

(312) APL/360 OS AND APL/360 DOS GENERAL INFORMATION MANUAL, DECEMBRE 1970,GH20-0850.

(313) APL/360 OS AND APL/360 DOS USER'S MANUAL,DECEMBRE 1970, SH20-0906

(314) APL/360 OS AND APL/360 DOS SYSTEM MANUAL,FIRST EDITION, JUIN 1971,LY20-0678.

(315) APL/360 DOS OPERATIONS MANUAL,SEPTEMBRE 1969,H20-0685.

(316) APL/360 DOS SYSTEM GENERATION MANUAL,SEPTEMBRE 1969, H20-0686.

(317) APL/360 DOS OPERATIONS AND INSTALLATION MANUAL,DECEMBRE 1970 SH20-0938.

(318) APL/360 OS OPERATIONS AND INSTALLATION MANUAL,DECEMBRE 1970, SH20-0890.

(319) APL/360 TYPE III PROGRAM DOCUMENTATION,REPORT 360D.03.3.007.

(320) APL/360 PRIMER,SECONDE EDITION,JANVIER 1970,GH20-0689.

(321) APL/360 USER'S MANUAL,MARS 1970,GH20-0683.

(322) APL/360 BENUTZERHANDBUCH,GH12-1030.

(323) APL AUDIO EDUCATION PACKAGE,3 VOLUMES,1971.
SR20-9382,SR-20-9383,SR20-9384.

(324) APL REFERENCE DATA CARD,S210-0007.

(325) APL KEYBOARD TABS,GX20-1783.

(326) INTRODUCTION TO MINIPERT,THIRD EDITION,MAI 1971,GH20-0852.

(327) APL CAI GUIDE AND PROBLEM BOOK,IBM SDD LABORATORY EDUCATION DEPARTMENT,ENDICOTT,N.Y.,SEPTEMBRE 1970.

(328) APL/1130 KEYBOARD TABS,GX20-1784.

(329) APL/1130 PRIMER,STUDENT TEXT,SECOND EDITION MARS 1969.
GGC20-1697.

(330) APL/1130 CONTRIBUTED PROGRAM LIBRARY,REPORT 1130-03.3.001.

(331) IBM MINIPERT,DATAMATION,1 MARS 1971,P.59.

(332) KEC.(KOMMUNERNES EDB-CENTRAL,COPENHAGEN)
MAO5 USER'S PROGRAM DESCRIPTION.
KEC-PERT

(333) INDUSTRIAL COMPUTER SYSTEMS=MINI-MANUAL,A CONDENSED INTRODUCTION TO APL,254 WEST 31 STREET,N.Y. 1001.

(334) PROPRIETARY COMPUTER SYSTEMS,INC.=PCS/APL PUBLIC LIBRARY GUIDE,FEVRIER 1971.

FMS CONCEPTS,1970.

(335) STSC=APL/360 PACKAGE GEARED TO SHARED LARGE FILES,COMPUTERWORLD,17 JUIN 1970,P.24.

(336) MISPAK-MANAGEMENT INFORMATION SYSTEM,APL PLUS.

(337) SUN LIFE MONTREAL,ACTUARIAL PACKAGE.

(338) THE COMPUTER COMPANY,RICHMOND,VIRGINIA=FMS-FILE MANAGEMENT SYSTEMS,1970.

(339) UNIVERSITY OF MARYLAND=APL/1100.

(340) XDS=APL/UTS,DATAMATION,NOVEMBRE 1971.

AN APL APPROACH TO INTERACTIVE DISPLAY TERMINAL GRAPHICS

W. H. Niehoff and A. L. Jones
IBM Corporation
Systems Development Division
P. O. Box 6
Endicott, New York 13760

ABSTRACT

Large, generalized graphics packages, as well as specialized graphic application packages, have not been especially successful in their penetration into the daily computing habits of computer users. We believe that this has been because of the relatively poor availability of display terminal equipment and limited useability of the programming support. An object lesson is provided by the acceptance of the APL language and its System/360 implementation. Its penetration into the working habits of users has been dramatic.

APL/360 GRAPHPAK* is an integrated collection of functions, implemented entirely within APL, that couples the facilities of APL/360 with economical, commercially available hardware to implement a highly interactive, easy to use graphic display facility. It attempts to employ the same attributes of APL that make APL attractive to yield a similarly pervasive system.

This paper will discuss the design philosophy behind GRAPHPAK, its basic functions, its application-oriented functions, and applications which have sprung from these basic facilities. It will attempt to show why this facility has demonstrated that useable, highly interactive, and economical computer graphics is very definitely possible in today's technical environment.

Introduction

APL/360 GRAPHPAK is an integrated collection of APL functions that was originally informally assembled to satisfy a need - the presentation of graphic information at the terminal of an APL/360 user. During 1969, the authors had searched for a means of presenting graphic data that was superior to the frequently-used APL typewriter plot packages. That search was successfully concluded with the discovery of commerically available plotter-controllers. A plotter-controller is inserted between the IBM 2741-Data Set interface where it monitors all serial data transmitted from a computer to the terminal. The plotter-controller's character translation and control conventions must be compatible with those of the TSP-12** (with erase feature). No modifications to equipment are necessary, and the terminal may continue to be used in the conventional manner. On receipt of a particular control character sequence, the plotter controller inhibits further transmissions to the terminal, and it buffers and digital-to-analog converts subsequent characters into analog deflection signals for a display device. Output devices used include storage tube displays and standard X-Y plotters (such as Tektronix Model 611 Storage Tube Display or a Hewlett-Packard Model 7005B X-Y Plotter. GRAPHPAK provides the programming support required to operate this equipment.

GRAPHPAK meets the objectives of a philosophy that strives to get computer graphics capability directly into the hands of the user. It meets at least four requirements of such a philosophy.

1. Economy - At current prices, the additional equipment required can be purchased for approximately $5000. This includes a plotter-controller, a storage tube display device, and a camera for hard copy.

2. Availability - The equipment is directly in the hands of the user - a part of the terminal he is using more and more in his daily working habits.

3. Useability - GRAPHPAK takes advantage of APL's conciseness and preciseness of notation. Graphic commands are a reasonable marriage of natural language and function notation. The result is that GRAPHPAK is easy to learn and easy to use.

4. Interactiveness - GRAPHPAK is highly interactive, primarily because of its availability and useability. Interestingly, the interactiveness is achieved in spite of relatively low performance, primarily because the manner in which pictures are developed maintains the interest of the user. The system is also highly interactive in that the user can interrupt a picture at any time while it is being drawn if he does not like what he is seeing.

Since its initial demonstration in early 1970, the facilities of GRAPHPAK have grown to emcompass a number of application areas. The author's attribute its growth to two factors:

1. APL, the language through which a user works with GRAPHPAK, makes it easy to implement new applications.

2. The ease of use of GRAPHPAK encourages, rather than frightens, APL users to add the graphics dimension to their work.

GRAPHPAK Facilities

GRAPHPAK consists of facilities of two types - basic graphics support and applications support.

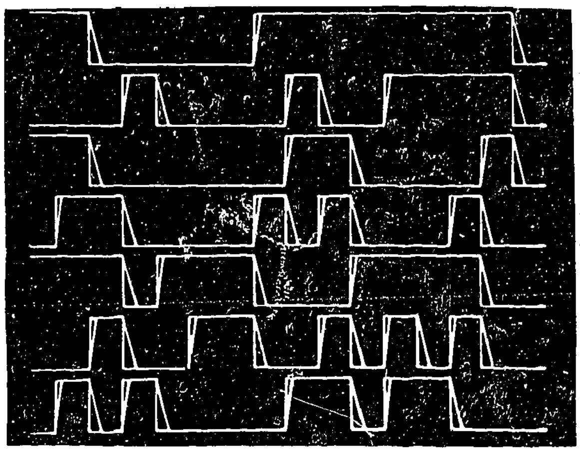
The basic support provides several simple, but non-trivial facilities.

1. It provides the ability to draw defined absolute vectors in a 0-to-511 x-y coordinate system.

2. It enables generation of stroked characters of varying size and orientation.

3. It allows the user to automatically erase the screen of a storage tube display.

An example of a display generated using the basic "DRAW" facility is the timing diagram illustrated below.

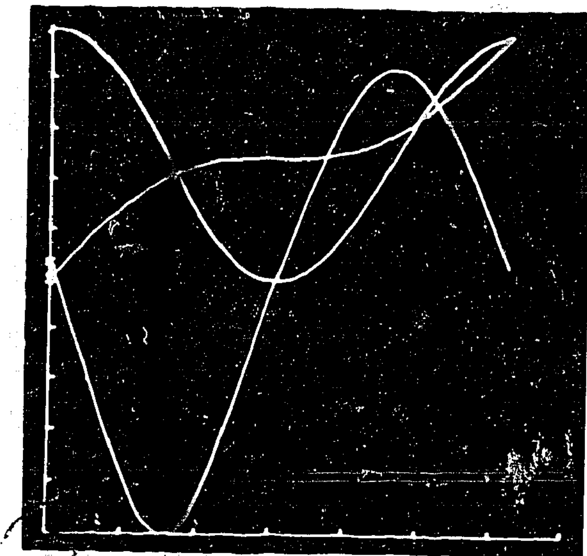


Character-writing, illustrated below, is generally to be avoided, since it is exceedingly slow. (Drawing proceeds at a speed of about four line segments per second.)

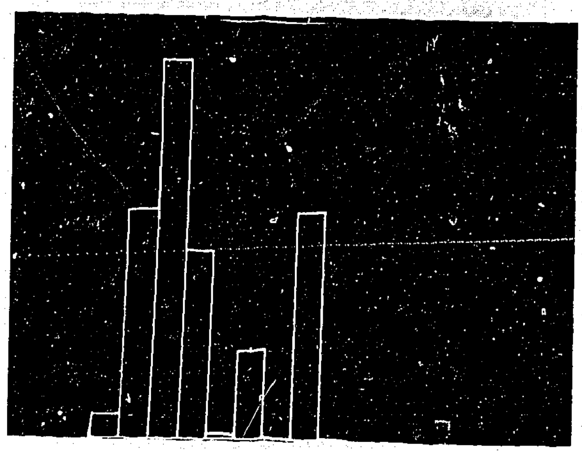
Applications support is built on the basic functions, and it includes functions for curve-plotting, curve-fitting, and descriptive geometry. Examples of each are illustrated on the following pages.

Curve-Plotting

A multi-function plot:

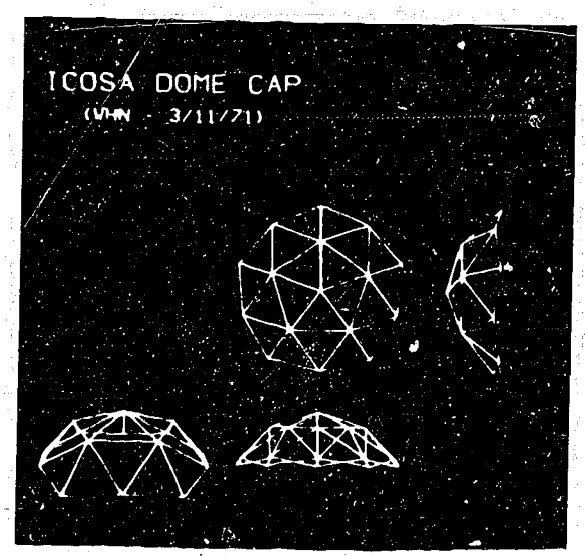


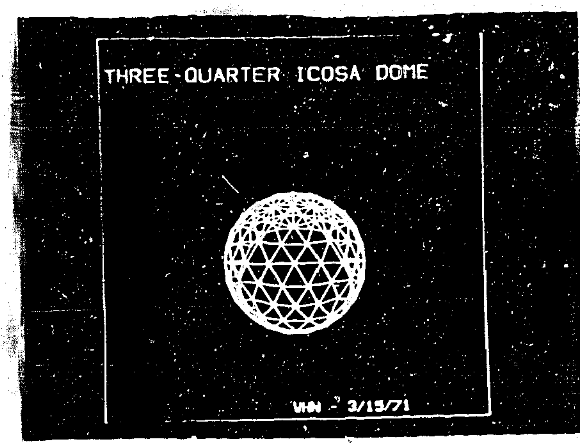
Barchart:



Descriptive Geometry

Icosahedon-based geodesic domes:

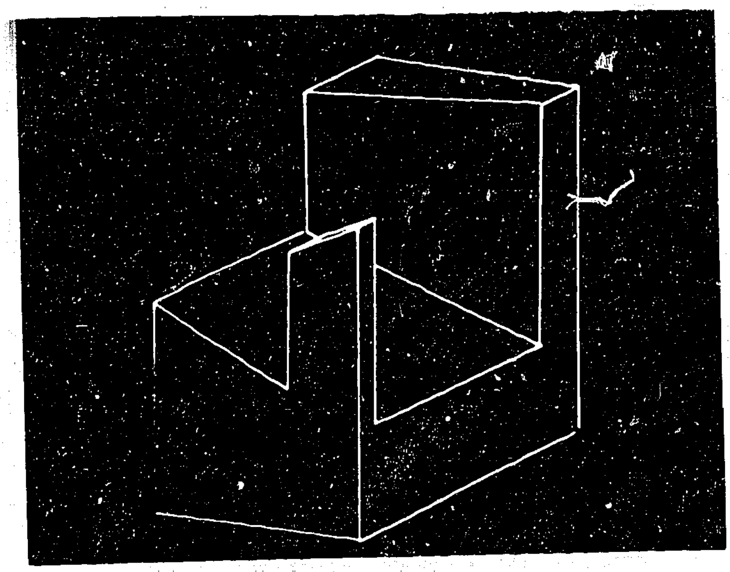




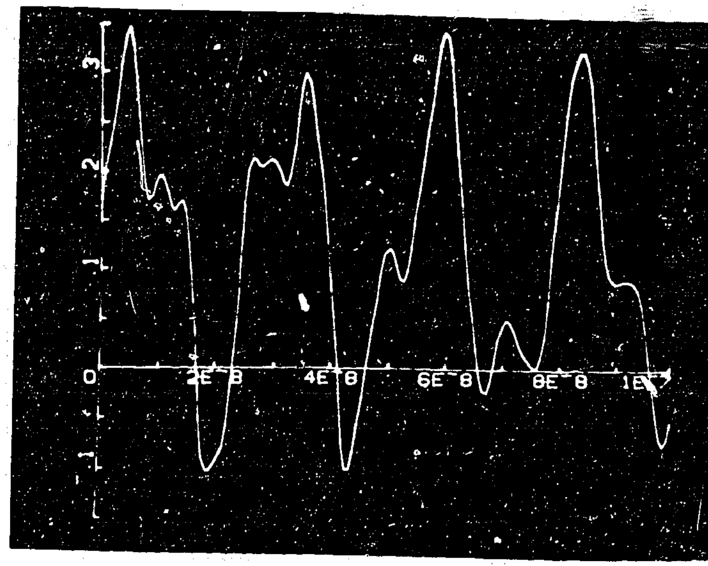
Actual Applications

Following the demonstration of the original GRAPHPAK package, individuals have built on the capabilities and have implemented their own applications. Examples are illustrated below.

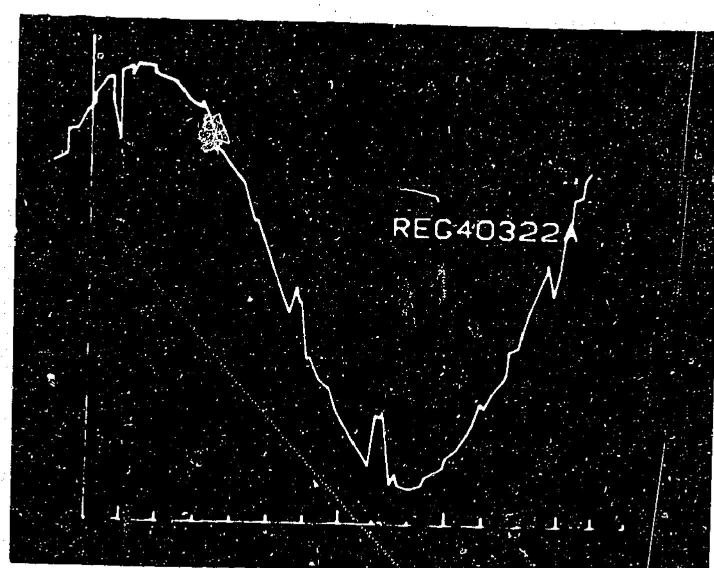
A graduate student has implemented a hidden surface removal algorithm in APL:



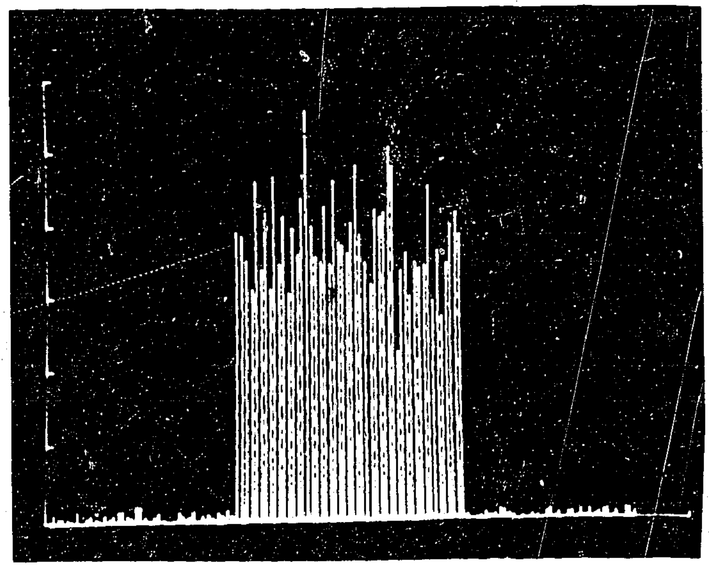
An electrical engineer has used GRAPHPAK to confirm proper reconstruction of Fourier-analyzed signals.

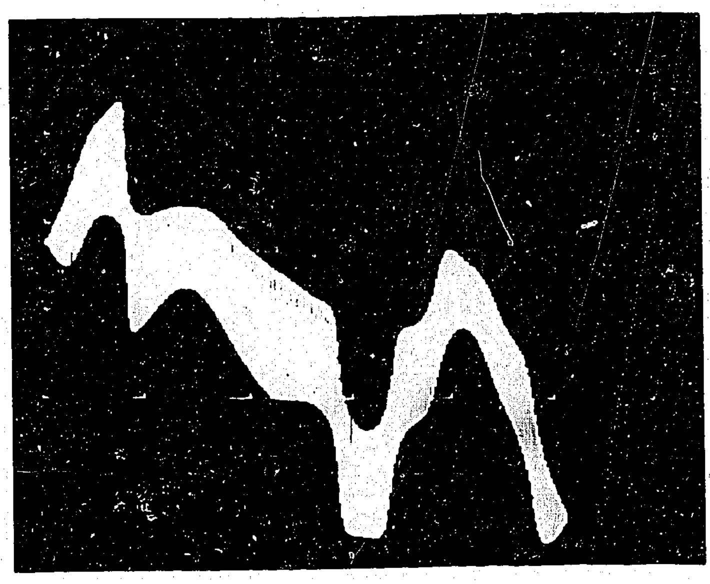


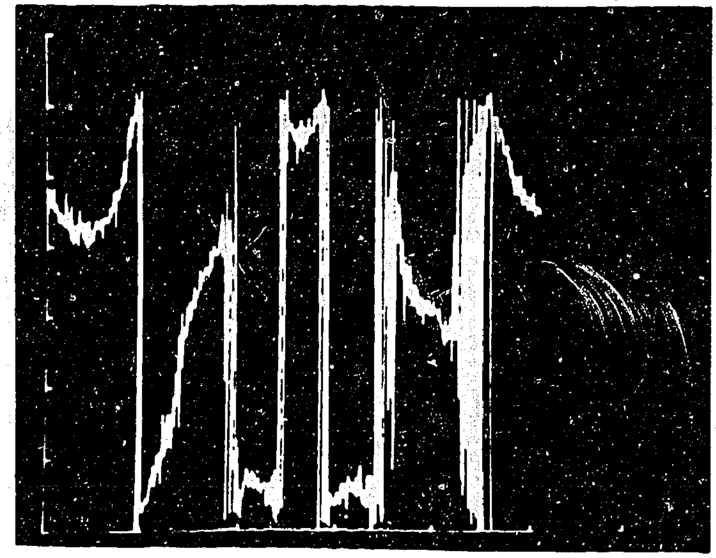
A physicist has used the plotting facilities for spectral analysis.



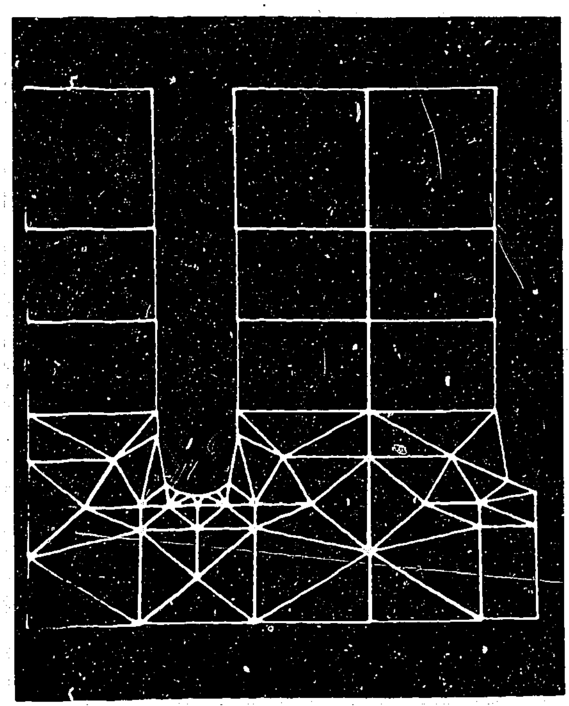
Optics specialists have used the plotting facilities in studies of kinoforms and optical filters:







An engineering mechanics specialist has used the sketching facilities to conform a finite element model:



Syntax

The authors feel that the APL syntax is ideal for a conversational graphics language. This is due to the manner in which several APL functions may be called with one line of typing in the desk calculator mode. For example, if a user has an object represented as a set of data, say *OBJECT*, he can draw a rotated, scaled, and translated perspective version of it on the screen with the command:

SKETCH ¯.6 ¯.6 0 *TRANSLATE* 2 *PERSPECTIVE* 20 30 40 *ROTATE SCALE OBJECT*

The syntax of some of the GRAPHPAK functions is given here:

Absolute Vector Drawing

Points in two-dimensional space(X-Y) are most basically addressed by specifying coordinates in the range 0 to 511. The APL function xxxxxx converts coordinates in this range into a string of APL characters (literal vector) which the plotter controller then converts into analog voltages used for driving a display or plotter.

SYNTAX: *Z←DRAW DATA*

WHERE:

DATA	is a three-column matrix
DATA[;1]	is a binary vector. A 1 means to go to the (x, y) position indicated by columns 2 and 3 with the beam off (or the pen up). A 0 means the same thing but with the beam on (or the pen down).
DATA[;2]	is a vector of X positions.
DATA[;3]	is a vector of corresponding Y positions.
Z	is the literal string (graphic orders) which will cause plotting when communicated to the terminal.

If a drawing is to be produced with the beam on (or pen down) for all points, the first column of DATA may be omitted. That is, only the X- and Y-coordinates need be included. When the function is used in this way, it is assumed that the pen should be up as it moves to the first point.

The function *DRAW* will generate output for use with a CRT display or with an X-Y plotter. When used with a CRT display, a line can be drawn from one side of the screen to the other with four characters. The time needed for this is about .27 second, because the characters are sent at the rate of 14.8 per second. However, when using a plotter, intermediate points must be inserted since the response time of the mechanical pen is much slower than the CRT. This is done by computing the size of the increments of motion required; then, if any of the ΔX or ΔY are larger than a preset value (usually 50 is used), extra points are inserted by linear interpolation so that all of the ΔX and ΔY are smaller than the preset value. If the pen is up, this interpolation is not done. However, after a long pen-up movement, the pen needs time to "settle down"; so the X, Y position is called twice to allow for the settling. After a string of data has been processed and the lines drawn, a variable called *LAST* is set which contains the coordinates of the last pen position. Then, when the next string of data is entered, the program knows the pen position and can decide if interpolation is needed. All of these features are bypassed if the output is on a CRT display. The global variable *DISP* is set to 1 or 0, depending on whether the output device is a CRT display or a plotter respectively. The interpolation distance may be altered by changing the variable *DIST* from its normal value of 50.

If a data value is outside the 0 to 511 range, one of two operations will take place depending on the value of the global variable *SCI*. If

SCI = 0, the data is changed to the nearest value in this range.

SCI = 1, the data is "scissored"*** to present a non-distorted object on the display.

Also, if a data value is an non-integer it is rounded to the nearest integer.

Character Writing

Characters are written using the function *WRITE*. The characters which may be written are:

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890⎕×○-⋆+.,/¯<≤≥>≠∨∧÷→←()[]\;:↑↓¨?~
⍳⌈_∇∆∘⊥⊤|∊⍴⍺'⊂⊃∩∪=⍵⌊

SYNTAX: *Z←P WRITE C*

WHERE:

C is the character string to be drawn; *P* is a vector.

P[1 2] = x-y position (in 0-to-511 raster units) of the lower left corner of the first character.

P[3] = character height. If 1, the character is six raster units high and four wide. If *P*[3] is a value other than 1, the size of the character is multiplied by this number.

P[4] is omitted if the characters are to be written horizontally. If *P*[4] is included its value gives the number of degrees that the line of characters is rotated counterclockwise about the about the point *P*[1 2].

Z = output string of graphic orders.

Curve-Plotting

The function *LPLT* takes data to be plotted and control parameters as its input and automatically produces scaled plots as output.

SYNTAX: *A LPLT B*

WHERE: *B* is an array containing the data to be plotted. If it is a vector, the values of the vector components are plotted against their index (i.e., *B* versus ⍳⍴*B*). If *B* is a matrix, the first column is considered to be a set of abscissa values and each succeeding column to be a set of ordinate values. Therefore, several sets of data can be plotted against a single set of independent variable values.

A is a vector of control parameters

$$A[1] = \begin{cases} 0 - \text{scale the data to fit within the plot frame} \\ 1 - \text{use the previously computed scale factors to scale the data} \end{cases}$$

$$A[2] = \begin{cases} 0 - \text{draw the axes} \\ 1 - \text{do \underline{not} draw the axes} \\ 2 - \text{draw axes only (no plot)} \end{cases}$$

$$A[3] = \begin{cases} 0 - \text{use a linear x-axis} \\ 1 - \text{use a logarithmic x-axis} \end{cases}$$

$$A[4] = \begin{cases} 0 - \text{use a linear y-axis} \\ 1 - \text{use a logarithmic y-axis} \end{cases}$$

$$A[5] = \begin{cases} 0 - \text{plot line segments between points} \\ 1 - \text{plot symbols on points} \\ 2 - \text{plot both lines and symbols} \end{cases}$$

$$A[6] = \begin{cases} 0 - \text{do not label the axes} \\ 1 - \text{label the axes} \\ 2 - \text{label axes only (no plot)} \end{cases}$$

Only the first component of *A* is required to be entered; it is automatically filled out with zeros to a length of six if components are missing.

If data falls outside the range 0 to 511, whether "scissoring" will or will not be applied is determined by the value of global variable <u>*SCI*</u> , as described earlier.

The terminal printer records zero-shift and increment values for both axes after each new plot unless labeling has been called for.

As a final example of the syntax used in GRAPHPAK, we consider the problem of fitting a smooth curve to a set of data, say *XYTEST* . If we decide we would like to see how a straight line fit would look, we enter:

FIT SL XYTEST

The system responds by drawing a graph of the data points and the least-squares best-fit straight line. Then we try a third-order polynomial:

FIT 3 *POLY XYTEST*

The system "knows" that it has already drawn the data points and the axes so it doesn't bother doing that again but proceeds to draw the third order polynomial. In addition to polynomial fits, GRAPHPAK has the capability to do power, exponential, log, log-log, and spline-like fits.

<u>Summary</u>

It has been encouraging to watch the growth of GRAPHPAK's acceptance. Today, it is being used in several IBM locations, including Endicott, Fishkill, Lexington, Los Angeles, Philadelphia, and Yorktown, and it appears that an applications base will be built in a manner similar to the way the APL public library base was assembled.

GRAPHPAK was recently announced as an IBM Field-Developed Program. We expect to find it applied to a wider class of applications, since, through its development and use, we have become convinced that useable, highly interactive, and economical computer graphics in the context of APL is very definitely possible in <u>today's</u> technical environment.

FOOTNOTES

* APL/360 GRAPHPAK, Program No. 5798 AGK-IBM Corp., Program Information Department, Hawthorne, New York.

** Time Share Peripherals Corporation, Miry Brook Road, Danbury, Connecticut 06810.

*** Scissoring is an operation that truncates image data to produce the illusion of cutting the image on the specified boundaries.

GRAPHICS IN APL

Alfred M. Bork
Physics Computer Development Project
University of California
Irvine, California

This document describes an experimental graphic facility within APL. The terminals are assumed to be inexpensive timeshared graphic terminals equipped with an APL character set. We first describe functions in a grpahic workspace, and then APL primitives for graphing.

User Plotting Functions - Workspace DRAW

The following functions are available as a group called SEE in the DRAW workspace:

DRAW	NOSCALE
TEK	CENTER
ARDS	SET
TERMINAL	INT
ERASE	VS
SCALE	DASH
AXES	

1. DRAW produces a curve on the screen and determines where the curve is to appear.

   It imitates, at least partially, the APL/360 PLOT function. PLOT produces point plots or histograms on the typewriter. Its general form is

   20 50 PLOT X VS Y

   VS is an APL function, combining the two vectors X and Y ($\rho X = \rho Y$) into an array suitable for use in PLOT. The numbers in front have an effect somewhat like windowing - they determine the "size" of the plot, the number of lines and the number of characters in each line.

   The corresponding graphic function is DRAW. It follows the general specification for PLOT. DRAW only plots <u>one</u> curve each time, in <u>either</u> two or three dimensions. It seems natural to let the left argument of DRAW specify a "window," a section of the screen on which the picture is to appear. It can use the function VS to combine arrays for plotting, or 1 by N, 2 by N, or 3 by N arrays can be used directly as the right argument.

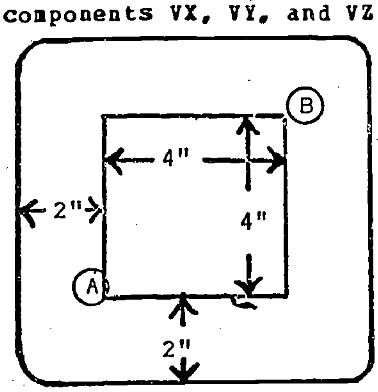
   We need four numbers for a window, the coordinates of points A and B in inches from the lower left corner, as in the diagram, so DRAW can be preceded by a four-vector, literals or a variable.

   If the left argument to DRAW is a scalar, the window currently in effect applies; the value of the scalar is ignored. The initial default window is the largest square possible touching the lower and right edges.

   DRAW can have a third argument on the right for three-dimensional plotting. Thus

   2 2 6 6 DRAW VX VS VY VS VZ

   plots the three velocity components VX, VY, and VZ in a window as shown:



We must have $\rho VX = \rho VY = \rho VZ$. VS is extended to allow 3-D arguments to DRAW.

The distances for the first arugment of DRAW are measured in inches from the lower left corner. After a DRAW, the cursor goes to the next writeable line. DRAW does not erase the screen, so it can be used to overplot curves.

2. SCALE, NOSCALE, and CENTER determine the placement of the picture within the window.

SCALE determines the <u>user</u> coordinates for the smallest and largest value, the corners of the current window. The general form is

SCALE A

For a 2D plot, A is a four vector; the first two components are the maximum and minimum values of the horizontal variable, and the next 2 of the vertical variable. For a 3D curve, $\rho A = 6$; the last two components determine the scale for the third, or Z, axis.

So for a 2D graph where the minimum values are -1.5, and the maximum 3, for both variables, we have

SCALE ¯1.5 3 ¯1.5 3

The default for SCALE is to scale the data to occupy the full window, finding the maxima and minima.

NOSCALE returns to this default case after the use of SCALE. It has no arguments.

CENTER places the origin of the coordinate system in the center of the windown, and then scales to fit the window. It has no arguments.

SCALE, NOSCALE, and CENTER do not return a value.

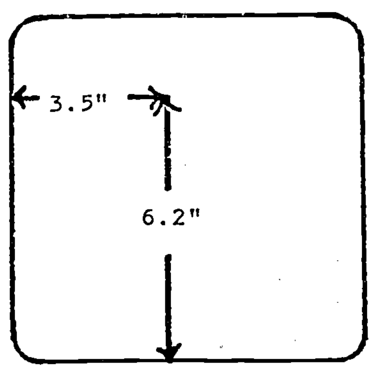
3. ERASE, HOME, and SET control utility functions on the CRT screen.

Screen control functions perform operations on the CRT, as in these examples:

ERASE - erases screen, sets cursor at upper left corner

HOME - sets cursor at upper left corner

3.5 SET 6.2 - the cursor is set to the position on the screen shown, with measurements in inches.



4. AXES draws axes corresponding to the current scaling and windowing conventions. It has no arguments, and returns no value.

5. DASH causes the <u>next</u> <u>curve</u> <u>only</u> to be dashed.

6. INT establishes an interval. It is often useful in plotting to establish a vector of equally spaced values. The function for this is INT, as in this example:

A ← INT -6 6 100

This function sets up a vector of 100 equally spaced values between -6 and 6, and assigns it to A.

7. ARDS, TEK and TERMINAL set the type of terminal in use. On initial release the APL Graphics facility supports three terminals: Tektronix 4002, Tektronix 4010, and ARDS

100. As these terminals have different graphic coding conventions, it is necessary for APL to know which is in use in order to draw curves.

In later versions of APL Graphic software terminal specification may use a system command. However the initial system employs the following functions to set terminal type:

> ARDS - sets terminal as ARDS
> TEK 4002 - sets terminal as Tektronix 4002
> TEK 4010 - sets terminal as Tektronix 4010
> TERMINAL - queries the user as to which terminal he is using, and takes appropriate action. Intended for use in graphics programs which do not suppose a highly knowledgeable graphics user.

The default terminal if no terminal is selected is the Tektronix 4010.

8. Later Features

Eventually we will allow the user to define <u>what</u> graphic terminal he is using, perhaps with a ")TERM" system command. This affects both code translation and graphic data.

We will also allow the user to "store" a picture, the actual graphic data; this may be done with a new data type, "graphic."

We will later allow for the possibility of graphic input, through tablet, light pen, joystick, mouse, etc.

DRAW may also eventually be called upon to construct functions in the complex plane, assuming that "complex" is defined as an APL data type.

<u>Underlying APL Primitives for Graphics</u>

1. $\boxed{0}$ - Quad backspace zero

This is the basic graphic output function. Its use is in the form

$\boxed{0}$ --- A

For ASCII terminals incapable of drawing APL characters the expression "$QO" can be used.

The following are legal possibilities for xxA:

1.	A = 2 N	2D plotting
2.	A = N 2	
3.	A = N	against indices
4.	A = 1 N	
5.	A = N 1	
6.	A = 3 N	3D plotting
7.	A = N 3	

This leaves several ambiguous cases. If ρA = 2 3, we interpret this as a 2D plot of 3 points. If ρA = 3 2, we understand a 3D plot of 2 points. If ρA is 2 1 or 3 1, a point is plotted. If ρA is 1 2, or 1 3, 2 or 3 points are plotted.

On initial implementation cases 1, 2, 6, and 7 are available.

Scaling, windowing, and the terminal currently in effect control the conversion of the arrays to graphic form. The graphic data is set to the terminal; the first bytes of data set the graphic mode and the last return to character mode. The screen is <u>not</u> erased by this operator.

This primitive is available and known to the user; the character $\boxed{0}$ is legal.

If a single number is assigned to ⎕, an ASCII control charcter is sent. The correspondence between integers and control characters is in ascending code order. Other As give a RANK ERROR.

At a later time ⎕ will be used for input, both for interrogating the terminal (as with the TEK 4010) and for graphic input from tablets, joysticks, mouses, etc.

2. ⌹S - Quad backspace S

Memory inset - for controlling graphic conversion. ⌹S transfers data entered with SETPOINT, SCALE, AND DRAW (window data) to the code for generating graphic data. We can do this with a command of the form:

⌹S C,A

where S is a new APL primitive, A is a scalar or vector, or a variable with scalar or vector value, and C is an integer specifying the function as follows:

C = 1	terminal type 1 = ARDS, 2 = Tek 4002, 3 = Tek 4010	
C = 2	lower left window, x lower left window, y, upper right window, x upper right window, y	ρA = 4 for C = 2
C = 3	scale, min x scale, max y scale, min y, scale, max y scale, min z scale, max z	ρA = 4 or 6 z arguments are optional
C = 4	setpoint - x + y values.	ρA = 2
C = 5	Draw axes	
C = 6	Controls scaling. If A = 1, use maximum scaling. If A = 2, use centered coordinates with maximum scaling. If A = 3, return to previously set window.	
C = 7	Dash next curve.	
C = 8	Erase screen	

Higher values of C presently give a Value Error; some of them may be used for future extensions. For terminals without APL characters, $QS can be used.

Examples:

1. Changing the window

   ⌹S 2 2 2 4 4

2. Scaling for coordinates, 2D plot

   ⌹S 3 3.4 7.1 4.1 6.3

⌹S is a legal character, but it is expected that it would not normally be employed directly.

The system is implemented in APL under the Universal Timesharing System for the Xerox Sigma 7. Implementation details are available in a separate document.

AN INTERACTIVE APL GRAPHICS SYSTEM

Stuart G. Greenberg and Craig I. Johnson
IBM Scientific Center
Cambridge, Massachusetts

ABSTRACT

An experimental APL graphics system operating under CP-67/CMS and communicating with an 1130/2250-IV is described. Features which make this system useful in interactive design are emphasized. As an example of the usefulness of the graphics system, an interactive plotting package is presented in detail.

I. Introduction

The purpose of this paper is to describe an experimental APL graphics system and to justify portions of the system design by presenting an application of the system capabilities. The two basic motivations for the development of an APL graphics system are the need for providing interactive graphics for APL and the need for providing computational power for an existing graphics system. The graphics routines and APL functions described in the paper are experimental programs for IBM internal use only and are presented in order to air the issues raised by the design of an interactive APL graphics system.

The major aim in the design of the APL graphics system was to achieve simplicity of use by the "non-programmer" while, at the same time, providing enough power to implement prototype complex graphics systems by the professional programmer. A secondary aim was to implement the system at a level such that the software would be adaptable to a wide range of display devices. Simplicity is achieved by the choice of the front end language itself. The conduciveness of APL to graphics applications is discussed in detail in Section 3. The achievement of the second aim is due to the fortuitous availability of existing hardware and software.

The first fortuitous circumstance was the development of APL(CMS) [1] a single user APL system running on a virtual machine under CP-67/CMS [2]. Effectively, each user of APL(CMS) has his own copy (and, in some cases, version) of the APL interpreter. Experimental facilities exist which enable the APL(CMS) user to execute external (to APL) object code. In particular, the user can execute the subroutine REMGRAF. REMGRAF is an interface between System/360 programs and the 1130 graphics support for the 2250 display terminal. The 360 and 1130 communicate via synchronous communication lines which are managed by a communications access method called HOTLINE. While the user never calls HOTLINE directly when using REMGRAF, the programming interface provided by REMGRAF makes available to the user routines which perform graphic functions on the 1130. The net effect to the user is the seemingly direct call to the 1130 graphics routines.

The restrictiveness of running under CP-67 is overcome somewhat by the ease of interface modification which follows from the modular structure of the graphics system itself. This modularity lies primarily in the descriptive data and command structures [3,4] - data structure, picture structure, and graphic "orders." A natural taxonomy of interfaces arises from the combinations of the potential places of residence for these structures. For example, the System/360 communicating with a 1130/2250-IV can make use of a problem data structure residing in the 360 while the picture structure and graphic orders reside on the 1130 (the system actually used). This separation of interfaces allows the implementation of compatible schemes for the use of different display terminals. For example, with a buffered 2250-I/III the data and picture structures would reside in the 360 and the graphic orders in the 2250 buffer. An unbuffered 2250-I (in effect, a storage tube) would require all structures to be on the 360 side and would further require the re-creation of the entire picture for each change. Thus, it is easily seen that the advantage of viewing graphics systems in this manner lies in its provision of a clear one-to-one relationship between interface location and display device capability.

The description below will relate to the dynamic display device (2250-IV) actually used in our experimental implementation. It should be pointed out that use of a different device does not require modification of the APL functions described in the sequel, but requires modification of the interface and communication code. The restriction on the APL functions lies in the fact that as the display device becomes less capable only a subset of the functions are applicable. The system actually used is pictured below in Figure 1.

For our purposes, the System/360 side consists of a CMS virtual machine. The CMS system code occupies the first 73728 bytes of virtual storage and is followed by the APL interpreter and execution code occupying approximately 50K bytes. REMGRAF and HOTLINE are loaded after this

and the remaining area (approximately 330K bytes on a 512K byte virtual machine) is the APL Workspace.

The APL graphics system described in this paper differs from the recently announced GRAPHPAK [5] primarily in its capabilities in dealing with a dynamic display such as the 2250 terminal.

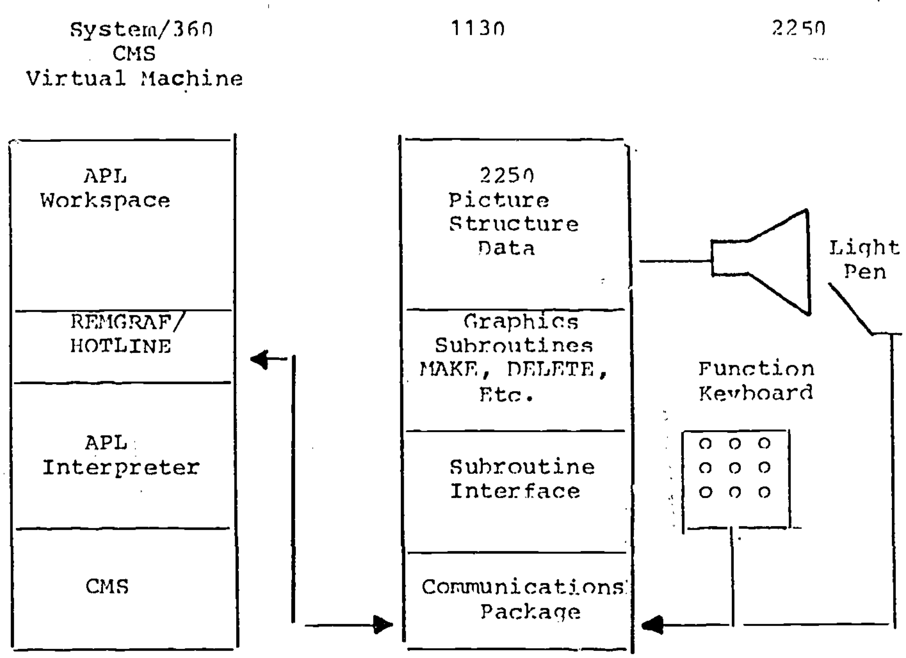


Figure 1 - APL Graphics System Configuration

The remainder of the paper will be concerned with the graphics system as it is viewed by the APL programmer. Section 2 contains a description of the basic graphics functions which are available. The advantages of the APL graphics system in interactive design are discussed in Section 3, and an application of the basic functions in an interactive plotting package is presented in Section 4. Extensions are discussed in the concluding Section 5.

II. The Basic APL Graphics Functions

The APL functions listed below correspond to the graphic command capabilities of the 1130/2250-IV system described in the preceding section. The communication interface may be completely ignored if desired. However, there exist commands which are useful for blocking and unblocking messages in order to improve performance. The commands fall roughly into five classes - initialization, entity creation and deletion, entity manipulation, transmission control, and offline device control.

Initialization

The commands are GRAPHICS and RESET:

GRAPHICS - initializes some global variables and the 2250 at the start of a graphics session

RESET - resets the 2250 display terminal

Creation and Deletion of Graphic Entities

The commands are MAKE, PTEXT, and DELETE:

MAKE - this command is used to display a set of coordinate pairs on the 2250. The format of this command as an APL function call is

Y MAKE X

where X is the set of abscissa points and Y is the set of ordinate points. There are several modes in which these points can be displayed. These modes are controlled by the global variable PLOTMODE which defaults to the integer value 7, meaning absolute lines. Using this particular mode results in lines connecting the specified points. Smooth curves would normally be displayed in this fashion. The other plotmodes are listed for convenience in Appendix A. The arrays X and Y should contain integers between 1 and 1024, these numbers corresponding to the actual raster units on the 2250 display terminal. The function MAKE returns an integer identification number by which one may refer to the graphic entity just created in future operations.

PTEXT - this command is used to display character strings on the 2250. The function call is

XY PTEXT STRING

XY is the coordinates of the starting point of the character string on the 2250, and STRING is the character string to be displayed. The display is wide enough to hold 74 characters. If a message goes off screen it simply wraps around to the next lower line. ID has the same meaning as in the function MAKE. Note that for both functions, MAKE and PTEXT, the normal id generation can be overriden. If one desires to assign a given plot or character string a specified id, then assigning this specified id to the global variable NID prior to issuing MAKE or PTEXT will achieve this desired effect. A particular instance where this might be useful is when one wishes to assign a group of plots and/or character strings a single id.

DELETE - this command enables one to delete graphic entities which have previously been created. The function call is

DELETE ID

ID is, of course, the identifying integer of the entity to be deleted. One may delete several entities at once, that is, the function DELETE may take a vector argument.

Graphic Entity Manipulation

The commands BLANK, UNBLANK, BRIGHTEN, UNBRIGHT, and READ are used to modify or manipulate the display of existing graphic entities.

BLANK - executing the command

BLANK ID

causes the named graphic entity to disappear from the 2250 terminal. The displayed image of the entity is stored in the 1130 memory, so that the entity may be redisplayed at a later time. ID may be a vector.

UNBLANK - this command causes blanked entities to be redisplayed. The function call is

UNBLANK ID

ID may again be a vector.

BRIGHTEN - this command causes the display of graphic entities to be brightened on the 2250 display. The function call is

BRIGHTEN ID

ID may be a vector.

UNBRIGHT - causes brightened entities to be restored to normal intensity. The call is

UNBRIGHT ID

ID may be a vector.

READ - causes a message to be read from the 1130. This message, which contains positional and id information, is initiated by light pen interaction with the 2250 terminal. The function call

A←READ

results in the assignment to the vector A entity id and light position information. Some particular uses of this function are described in Section 4.

<u>Transmission Control</u>

The commands BLOCK, UNBLOCK, and ENDBLOCK control messages to be sent from APL to the 1130

BLOCK - causes all graphics commands to be blocked from transmission to the 1130. These commands are stored in a buffer on the 360 side until a later time when the blocking is terminated and all the messages are sent. In effect, many changes may be made to the "picture" without these changes being reflected to the 2250. This could be considered the so-called "delayed mode" [6].

UNBLOCK - causes the blocked mode to be terminated. All messages which were buffered while in the blocked mode are now sent. The unblocked mode is the default operating mode and corresponds to the so-called "movie mode" [6], that is, each change to the picture is immediately reflected to give a dynamic view of the process.

ENDBLOCK - causes all messages buffered while in the blocked mode to be sent. This command differs from the unblock command in that the blocked mode remains in effect.

<u>Offline Device Control</u>

The commands PUNCH, READER, and PLOTTER control offline input/output devices

PUNCH - this command causes the image on the 2250 terminal to be punched onto cards by the 1442 punch attached to the 1130.

READER - this command causes a previously punched deck to be read by the 2501 card reader attached to the 1130. The stored image will be displayed on the 2250 terminal.

PLOTTER - this command causes the image on the 2250 terminal to be plotted on a Calcomp plotter attached to the 1130. This is a means of obtaining hard copy of graphics results.

III. <u>Usefulness for Interactive Design</u>

In addition to the aims of simplicity of use and adaptability to various display devices as discussed in Section 1, the APL graphics system described in the previous section possesses three additional important properties - it can exist with an independent data structure, it is serially flexible, and it is extensible. These properties make the system ideal for use in interactive design applications.

The independent data structure alluded to is that which was discussed briefly in Section 1 in reference to interface classification. The significance of the data structure goes beyond this, however. It represents, in the programmer's terms, a structural model of what the pictures displayed on the 2250 actually mean. This is important in simple applications and essential for

interactive picture manipulation. The functions of Section 2 make use of data which has already been converted to meaningful units for the 2250 terminal. The APL programmer has complete freedom to determine the source of his data and how it should be converted. Moreover, the APL programmer can build as complete or incomplete a description of the picture data as he desires. The data editing and computational capabilities of APL can be used to modify the data structure without necessarily changing the picture simultaneously.

By serial flexibility of a system we mean that the system provides the capability to proceed in a step-by-step fashion with each step thoroughly verifiable and correctable. The user of the system should be able to back up to any step he has been through and proceed once again from that point. By saving the active workspace at appropriate times, the user of the system may try completely recoverable alternative approaches.

The concept of extensibility is really a feature of the APL syntax itself and constitutes a justification for the use of APL as the front end to an interactive graphics system. With very little effort the APL programmer can package at any level, that is, application systems can easily be written for the entire user spectrum up to and including non-APL users. Clearly, APL is not unique in this, but its interpretive nature and the richness of its operator structure make the packaging an easier task. The price is paid, of course, in the relatively low execution speed of APL, a price which could be prohibitive in interactive design processes which require heavy, repetitive computations. One fortunate aspect of the APL graphics system is that the moderately costly command formatting and communications are not interpreted, but involve the execution of object code. In effect, these often used procedures have been put almost at the APL operator level.

The exemplification of these features in an interactive plotting package is contained in the following section.

IV. Interactive Plotting Package

An interesting example of the usefulness of the APL graphics system is the development of an interactive plotting package for the 2250 terminal written in APL. Using APL to write a plotting package enables one to exploit a natural language syntax which one could achieve in most other languages only by writing a special purpose interpreter. The plotting package has the characteristics mentioned in the preceding section - an independent data base, serial flexibility, and extensibility.

Briefly, the user of the plotting package creates and deletes plotting windows, makes plots within these windows, moves plots from one window to another, and puts labels and axes on graphs. In addition, some interactive capabilities are provided.

Plotting windows are created by a command of the form

GRAPH1←WINDOW 0 0 500 500

which will result in the display of a box with lower left-hand coordinates (0,0) and upper right-hand coordinates (500,500). The APL function WINDOW creates an integer identifier for the box and returns this as its output value. In the example above the output of WINDOW is assigned to the variable GRAPH1. One may then use the appellation GRAPH1 to refer to this window and its contents in the future. In addition to display and identifier creation, execution of the function WINDOW results in additional entries in the data base, most importantly, the addition of a new row in the matrix which is used to keep track of the various windows and their contents.

After one or more plotting windows have been created, plotting may begin by issuing, for example, the command

PLOT1←PLOT ((COS PI×T) VS T ON GRAPH1

where T is a preassigned vector of values ranging from 0 to 1 in increments of 0.02. This command results in the display of a half wave of the cosine function which is automatically scaled to fit within the window GRAPH1. The unique identifier assigned by the PLOT function is placed in the appropriate row of the window data matrix. This value is also returned as the output of PLOT, and, in this case, is assigned to the variable PLOT1.

THE APL function PLOT is more versatile than the above example might imply, and may be used to create multiple plots, to overlay on existing plots, and substitute for existing plots. The command

PLOT2←PLOT (FN1 T),(FN2 T), ... ,(FNK T) VS T ON GRAPHJ

results in the display of the appropriate functions on GRAPHJ. If there are already plots in window GRAPHJ, they will be rescaled and redisplayed with the new plots. The APL variable PLOT2 will be a vector of integers corresponding to the new plots. The command

PLOT2←PLOT (FN11 T),(FN21 T),(FN31 T) VS T FOR PLOT2

issued after the preceding command results in the new plots being substituted for the old plots denoted by PLOT2. Note that the plotting window need not be specified if it was the last window referenced.

The function MOVE is related to PLOT. It causes plots on one graph to be moved to another graph. An example of its usage is

MOVE PLOT2 TO GRAPH1

Note that usage of the functions VS, ON, FOR, and TO adds to the natural language syntax of the command structure.

Created entities may be deleted from the display terminal as well as the data base by using the ERASE function

ERASE NAME

where NAME is an integer constant or variable corresponding to a window or a plot. If a window is designated, the entire graph is erased. Erasing a plot will not cause a rescaling of the graph from which it has been erased. This can be achieved by issuing

REMAKE GRAPH

where GRAPH is the name of the window in which rescaling is desired. One other command, CLEAR, causes all of the plots in a window to be erased without erasing the window itself.

It should be obvious that all of the above commands make liberal use of the basic graphics commands MAKE and DELETE. The other content of these functions is primarily code associated with data base manipulations.

The default coordinates for plotting are cartesian. The mode of an existing graph called NAME may be changed to semilog or loglog by issuing

SEMILOG NAME

or

LOGLOG NAME

all further plotting associated with the window NAME will be in the appropriate mode. The mode may be returned to cartesian by issuing the cartesian command.

The display mode for a plot can be modified as well. The default mode is LINEMODE, but point plots or dashed line plots may be obtained by issuing

POINTMODE NAME

or

DASHMODE NAME

where NAME is a plot identifier. The plot will not immediately be changed, but the next REMAKE, PLOT, or MOVE associated the plot will cause it to be displayed in the appropriate mode.

Polar plots may be obtained by issuing PLOT POLAR instead of PLOT in addition to the usual syntax.

Labeled axes may be generated for a graph by issuing the command

AXES GRAPH

where GRAPH is a window identifier. The mode of the plot (cartesian, semilog, or loglog) is taken under consideration by the AXES function. Text labeling can be done by using the basic graphics function PTEXT.

Limited interaction with a light pen is provided to enrich the basic capabilities of the plotting package. All of the interactive functions involve the transmission of a simple message from the 1130 to the 360 as described in Section 2 under the function READ. These transmissions are initiated by the pressing of function key 10 while pointing at a graphical entity and pressing function key 10 again to terminate. The significant information contained in the message are the entity pointed at and the x-y position of the light pen when function key 10 is first pressed and the position of the light pen when function key 10 is pressed to terminate. The interactive functions which have been implemented are ENTITY, POINT, and SHIFT.

The function ENTITY READs the message sent from the 1130 and returns as output argument the integer identifier of the graphical entity pointed to by the light pen. An example of the usefulness of this function is the movement of a plot from one graph to another in the case where the identifier of the plot has not been conveniently recorded. One may then command

MOVE ENTITY TO GRAPH2

and then point at the appropriate plot with the light pen. Note that the command may be issued before or after the message is sent from the 1130.

The function POINT is used to provide the coordinates of a point on a plot. The coordinates are converted and given in the original data units of the graph.

By using the function SHIFT one may move textual entities to any desired position. The textual entity is pointed to at the first pressing of function key 10, and the desired position is pointed to at the second pressing.

The interactive functions described above are not meant to provide an exhaustive set of interactive capabilities. Indeed, much of the limitation on the interactive capabilities is due to the limited structure of the message transmitted from the 1130. Eliminating the existing constraints, however, would involve 1130 programming. This is not meant to imply that hopeless limitations exist. As a matter of fact, the property of extensibility embodied in the plotting package can lead to interesting generalizations. For example, one might wish to obtain detail in a given plot, that is, display some portion of the plot. This could be achieved by using the syntax

DETAIL PLOT PNAME BETWEEN POINT AND POINT ON GNAME

where PNAME is the plot which we wish to see in detail and GNAME is the window in which the expansion is to be displayed. Clearly, the function BETWEEN must extract from the data base that portion of the data associated with PNAME which lies between the two points indicated with the light pen. It is not difficult to see that this major extension of capability requires only a minor extension in code, namely, the writing of the simple functions BETWEEN and AND.

V. Conclusions

In order to reinforce the conclusion that the APL graphics system is useful, let us note some recent exploitations of the system.

Comba[7] has made an experimental implementation of his three-dimensional geometry language using the APL graphics system and an APL model of a relational data base. Lorie[8] uses an existing relational data base system (RAM) as an extension to APL(CMS) in addition to the APL graphics system in order to develop an experimental prototype mapping system. An interactive design system for transportation guideway optimization was done as a demonstration project for the U.S. Department of Transportation by personnel of the IBM Cambridge Scientific Center and Federal Systems Division. Another interesting exploitation was that done by the IMB Los Angeles Scientific Center in the area of three dimensional sculptured surfaces. The programs were originally written to display their results on a Computek storage tube. The APL graphics system was flexible enough to allow the conversion to a 1130/2250 configuration with two days of work by one programmer.

If anything philosophical can be drawn from the above described applications it is that the APL graphics system is a very useful system in which to "breadboard" prototype graphics applications. In addition, the flexibility in defining hardware interfaces adds a significant dimension in that each prototype system created is valid for many different hardware configurations.

APPENDIX A - USEFUL DISPLAY MODES

The following modes of display may be obtained by appropriately specifying the global variable PLOTMODE:

1 - incremental points

3 - absolute points

5 - incremental lines

6 - short absolute lines

7 - absolute lines ... default

10 - absolute lines beam off for odd coordinates ... dashed lines

15 - absolutely positioned large characters

26 - absolutely positioned basic characters

REFERENCES

1. APL(CMS) Program Description and Operations Manual, Document No. SH20-1088-0, IBM Corporation, February, 1972.

2. CP-67/CMS User's Guide, Document No. GH20-0859-1, IBM Corporation, September, 1971.

3. Johnson, C.I., "Fundamentals of an Interactive Graphics System," Preprints of the Symposium on Interactive Computer Graphics, Delft, the Netherlands, October, 1970, pp. 43-58.

4. Cotton, I.W. and Greatorex, F.S., "Data Structures and Techniques for Remote Computer Graphics," Proceedings of the Fall Joint Computer Conference, 1968, pp. 533-544.

5. GRAPHPAK - Interacting Graphics Package for APL/360, Document No. GB21-0411, IBM Corporation, March, 1972.

6. Johnson, C.I., "Interactive Graphics in Data Processing: Principles of Interactive Systems," IBM Systems Journal, Vol. 7, Nos. 3 and 4, 1968, pp. 147-173.

7. Comba, P.G., "A Language for Three-Dimensional Geometry," IBM Systems Journal, Vol. 7, Nos. 3 and 4, 1968, pp. 292-307.

8. Lorie, R.A. and Symonds, A.J., "Interactive Problem Solving Using a Relational Data-Base in APL," Digest of the 1971 IEEE International Computer Society Conference, Boston, Massachusetts, September, 1971, pp. 191-192.

MODELING A SATELLITE EXPERIMENT ON APL

Charles D. Wende
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This is a study of the charged particle measurements experiment (CPME) which will be flown on the IMP-H and IMP-J satellites. This experiment, although primarily intended to measure charged particles, contains detectors which are also sensitive to solar and galactic X-rays. Considerations aimed at optimizing the resulting X-ray data will be discussed, followed by a description of the technique used to unfold the intensities of individual X-ray stars from the data. Dummy data were generated from published X-ray star catalogs and used to test this analysis technique. Of particular interest to APL users are the storage/ retrieval of data packed as binary words of arbitrary length (to save room in core) and the programming, in APL, of the multiple linear-regression technique described by Bevington in his *Data Reduction and Error Analysis for the Physical Sciences*.

Introduction

This paper reports a simulation study of an experiment which will be flown on a satellite later this summer. As a result of this simulation, some refinements were made in the hardware, and more refinements will be incorporated in a second version which will be flown next year.

The experiments in question are the charged particle measurements experiments (CPME) on the IMP-H and IMP-J satellites. These satellites will be placed in orbits around the earth with perigees of 35 earth radii and apogees of 39 earth radii (i.e., about half way to the moon). They will be spin stabilized with their spin axes perpendicular to the ecliptic plane. As the satellite spins in this orientation, the sun moves along the satellite equator. The point of reference on the satellite equator will be the direction of the sun. This satellite point of reference will precess about one degree per day in a celestial coordinate system.

The CPME package contains five Geiger tube detectors and a set of three solid state detectors. These detectors are intended to provide measurements of charged particles in interplanetary space, although the Geiger tubes are also sensitive to solar and galactic X-rays. It is this secondary use of the Geiger tubes which will be addressed further.

The characteristics of these Geiger tubes are tabulated on Figure 1. The 'thick' and the 'thin' tubes both have relatively small window areas, about 1/25 sq. cm., and will be used primarily for solar observations. The 'big' tube has a large window relative to the other Geiger tubes and will be used primarily to observe cosmic X-ray sources. Note also that the Geiger tubes are sensitive to progressively higher photon energies.

The configuration of the CPME package is shown on Figure 2. Of particular concern are the three Geiger tubes oriented perpendicular to the spin axis. Two other thin tubes are oriented parallel and anti-parallel to the spin axis, and they comprise the North-South telescope. As they do not have enough sensitivity to observe cosmic X-ray sources, as they will not view the sun (if all goes well), they will not be treated further. The solid state detectors are insensitive to X-rays, and also will be disregarded.

Detector Selection

The first problem was to match the sensitivities and dynamic ranges o the Geiger tubes to the expected X-Ray fluxes. Since this experiment is not a pioneering experiment, some a priori information was known about the solar and cosmic X-ray spectra. Representative spectra are presented on Figure 3. Three solar spectra are given. One is a spectrum from a solar flare, another is from an active region which spawns flares, and a third is from the quiet sun background. These spectra vary with time scales of minutes, weeks, and years, respectively. Cosmic X-ray spectra shown include Scorpius, the strongest known X-ray star, Cetus, a variable source which was observed only once, Taurus (the Crab Nebula), which exploded as a supernova in the year 1054 A.D., Virgo, also called M-87, and the diffuse component, a background haze which is omnidirectional, here integrated over one steradian.

The expected count rates were computed by multiplying the photon spectra, at each photon energy, by the Geiger tube efficiency (i.e., the probability that a photon would cause a count to be registered), summing, and then multiplying by the window area to get counts registered by the tube rather than counts registered per square centimeter. Unfortunately, the calculation of the Geiger tube efficiency is complicated by its dependence on the type and thickness of

NAME	---- WINDOW ----		-- GAS --	PASSBAND	APERTURE	---- COMMENTS ----
	area sq.cm.	thickness mg/sq.cm.	thickness mg/sq.cm.	KeV.	deg.	
Thin	.04	0.35 Mica	0.34 Ne.	0.76-2.8	45 cone	Solar; 3 tubes (only one used)
Thick	.04	1.75 Mica	0.66 Ar.	1.04-9.8	45 cone	Solar
Big	.81	1.75 Mica +9.38 Be.	2.34 Kr.	1.46-14.5	40x11.25	Cosmic; IMP-J apert. 40x22.5 deg.

FIGURE 1. Characteristics of IMP-H/IMP-J Geiger Tubes (particle data excluded).

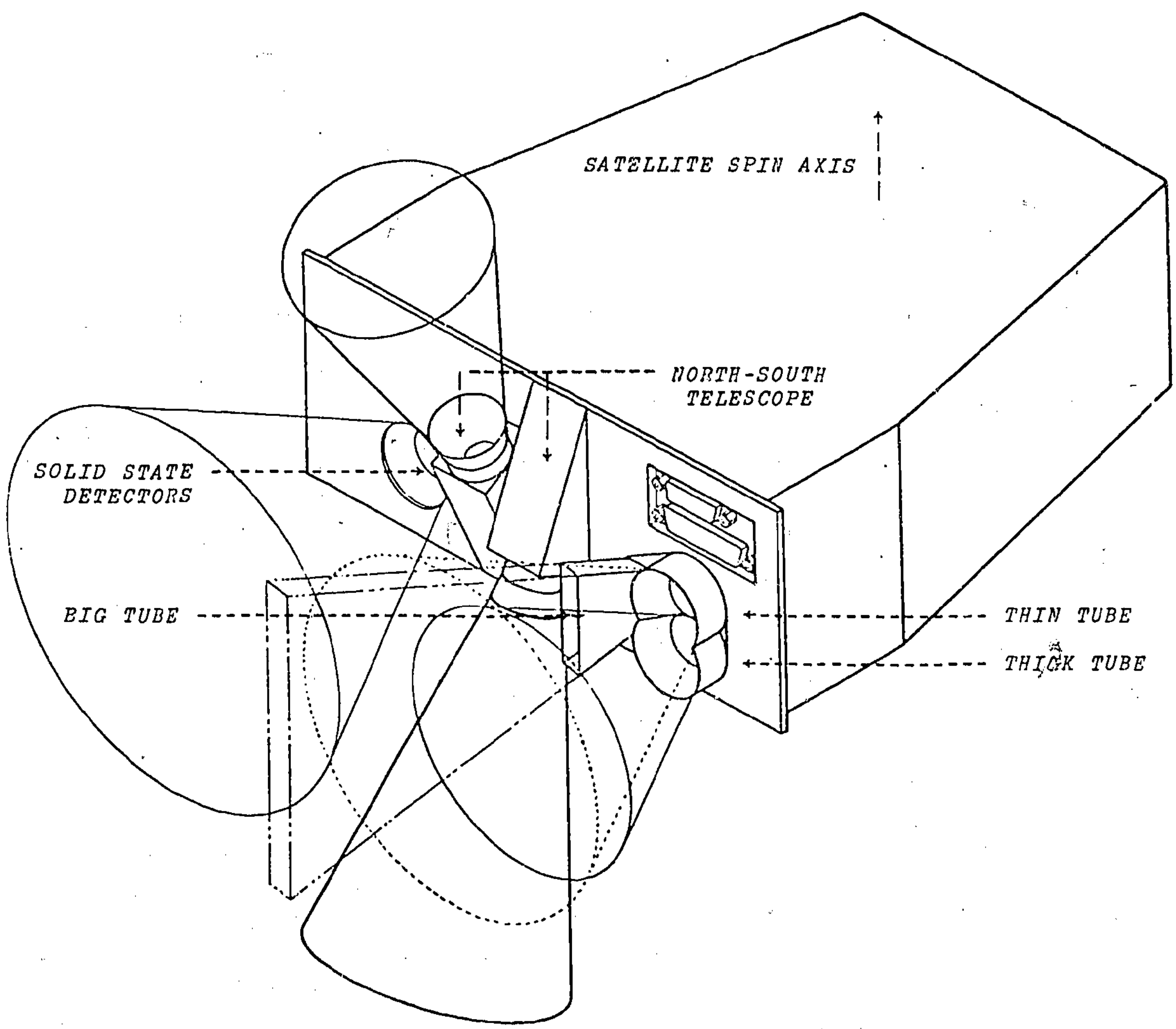


Figure 2. The Charged Particle Measurements Experiment Package.

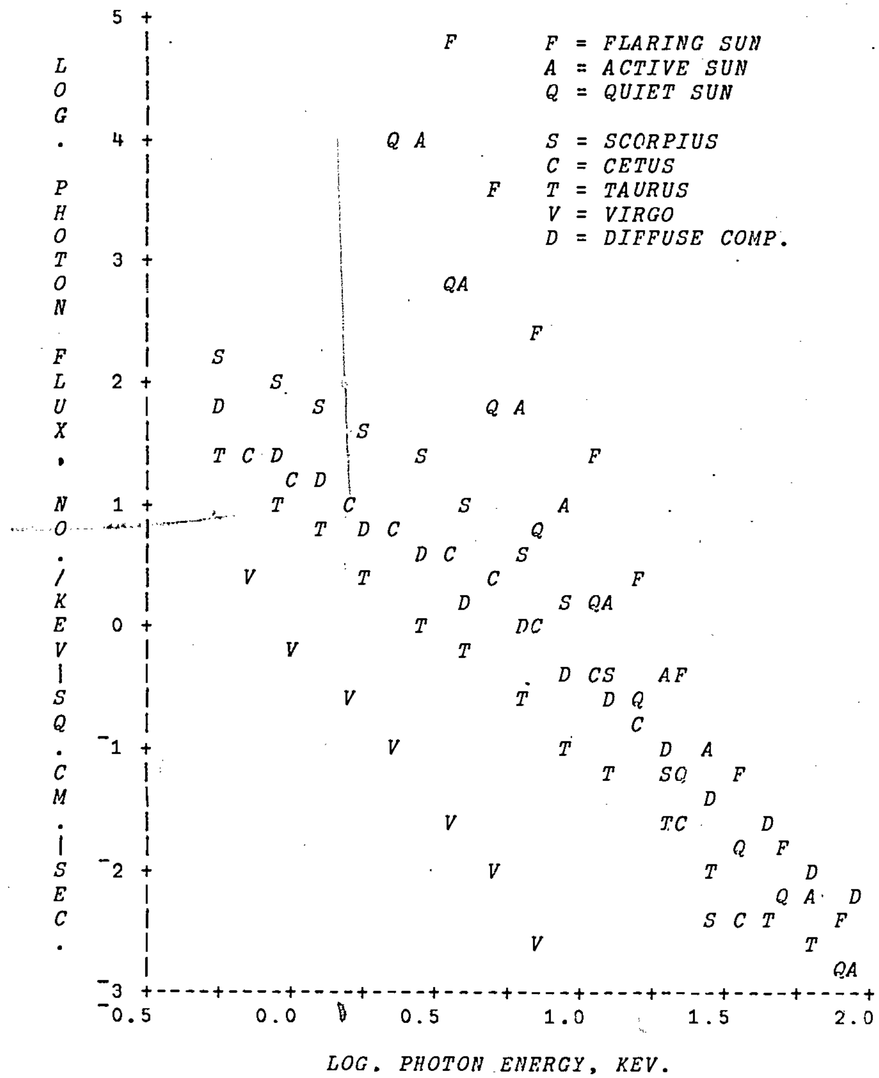


Figure 3. Solar and Cosmic X-Ray Spectra.

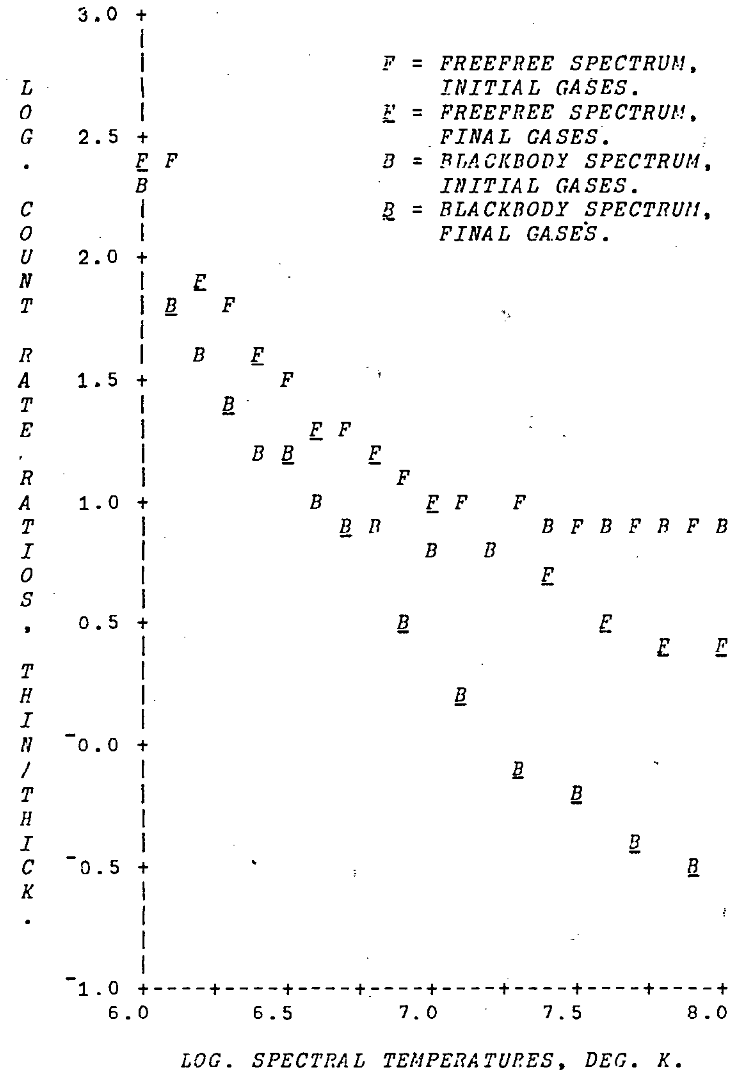


Figure 4. Count Rate Ratios Versus Spectral Temperature.

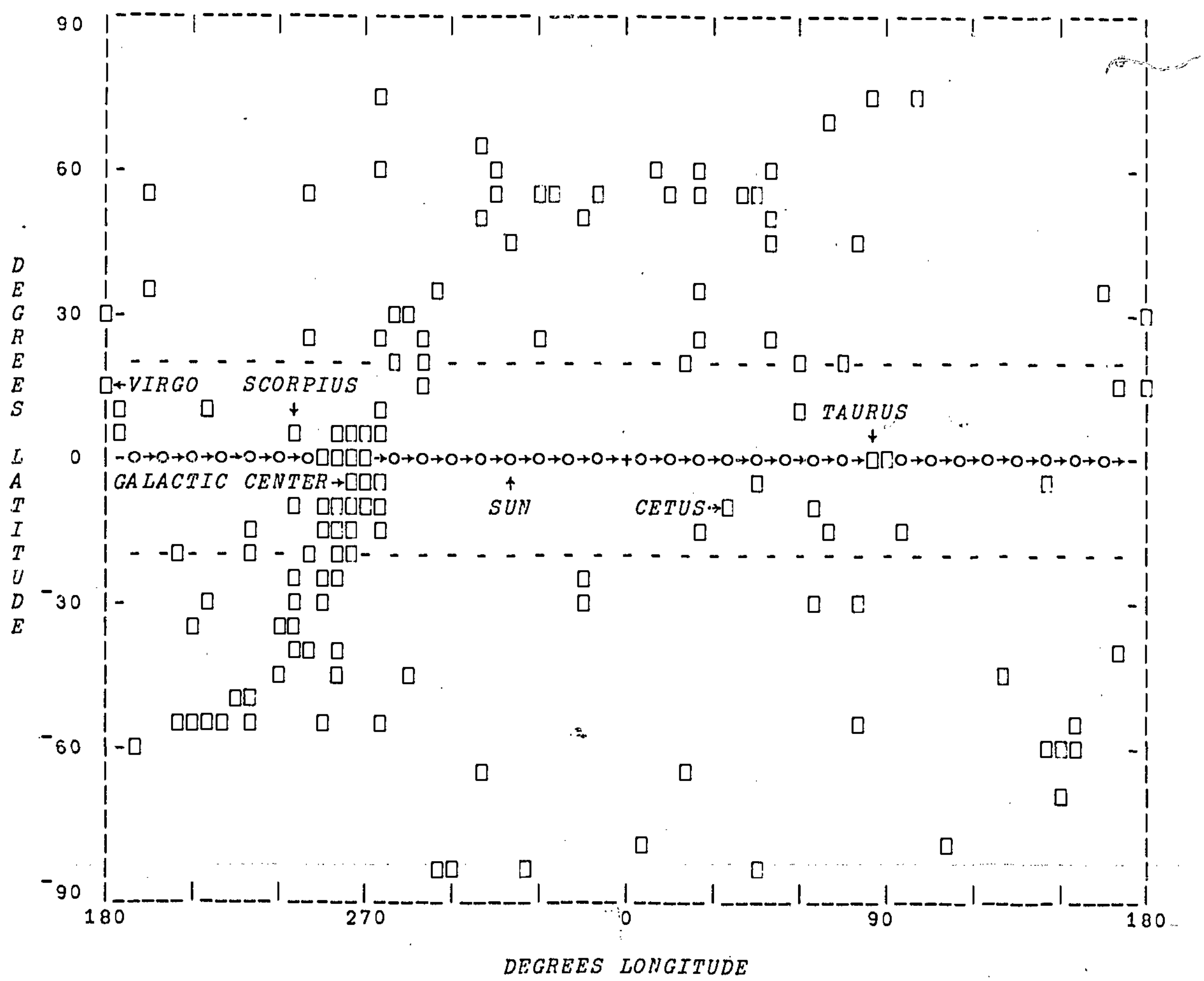


Figure 5. Cosmic X-Ray Sources in Ecliptic Coordinates.

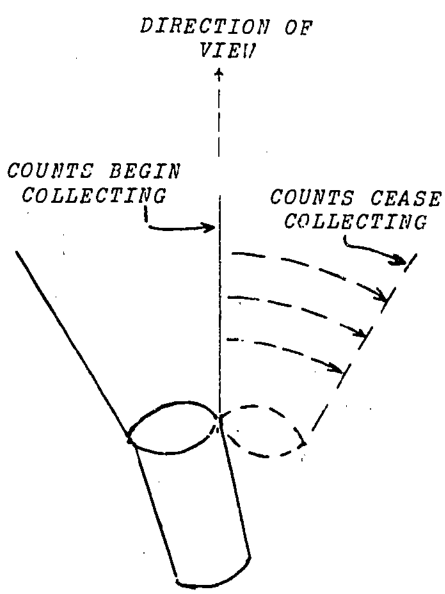


Figure 6. Accumulation Sequence Definition.

material in both the Geiger tube window and the filler gas. Further, the absorption coefficients needed vary logarithmically with the material type and photon energy, and also exhibit strong discontinuities due to the atomic structure of the materials. Normally the needed efficiencies would be calculated by plodding through massive tables of absorption coefficients and then grinding through needed additional calculations. Using APL, the tables were entered once, and a short interpolation routine enabled coefficients to be calculated for any wavelength in the region of interest. The calcuation of Geiger tube efficiencies was reduced to a few lines of APL. The calculation of expected count rates was also reduced to one line of APL.

When computing the expected count rates, it was found that the sun would probably saturate the thin tube. It was important not to compromise either the low energy response to solar X-rays or the low energy response to electrons. These constraints ruled out the simple solution of placing a foil, such as beryllium, in front of the tube. This problem was solved by placing a copper strip, perforated with holes over one per cent of its area, across the aperture in front of the thin tube, and orienting this strip parallel to the equator of the satellite. Thus, the count rate due to solar X-Rays would be decreased by a factor of 100 while the X-ray passband still extended down to 0.75 KeV, and the aperture would, for the most part, remain clear for the entrance of 15 KeV electrons. The thick tube, having been flown before, did not have this problem. The big tube, however, would saturate during solar flares but probably not during non-flaring time periods. This handicap was accepted in order to maintain the sensitivity needed to observe cosmic X-ray sources.

Another brief exercise that resulted in a small but significant change was the following. The ratio of the count rate of the thick tube to the count rate of the thin tube was calculated for various spectra. For historical reasons, initially the gas fills of these two tubes were reversed. The resulting ratios are shown on Figure 4. It was found that the original combination of gas fills produced ambiguous results. That is, the thin tube counted higher at low temperatures dur to the thin window, but also counted high at high temperatures due to the argon gas fill. Reversing the gas fills resolved the problem.

The above exercises are not elegant examples of APL coding, but prove to be extremely tedious to do otherwise. The use of APL allowed many different combinations to be tried and the optimum picked with very little expenditure of time.

Unfolding X-Ray Star Intensities

The remainder of this paper will be devoted to the technique developed for analyzing data from the big tube.

The X-ray sky, as viewed from IMP-H or IMP-J, is shown on Figure 5. This coordinate system is fixed in celestial space, however, rather than on the sun. The sun will move along the equator as time progresses. The aperture on the big tube limits the field of view to within plus or minus 20 degrees of the ecliptic equator (note the dashed lines). This segment of the sky does include many strong X-ray sources, such as Scorpius and the clump of stars at the center of our galaxy. One can determine which sources should be observable to the experiment and can limit the number of sources used in further modelling of the experiment.

The manner in which counts are accumulated is illustrated on Figure 6. As the satellite rotates, when the leading edge of the aperture, or collimator, is pointed in a given 'direction of view,' counts begin collecting in an accumulator; when the trailing edge of the collimator has rotated so that it points in the direction of view, counts stop collecting in that accumulator and begin collecting in the next accumulator. On IMP-H there are 32 of these directions of view, the first one offset from the sun by 10 degrees (i.e., the satellite rotation is divided into 32 sectors). On IMP-J, there will be 16 sectors rather than 32.

In determining the expected count rate from X-ray stars, the aperture function used is not simply the off-axis area relative to viewing the aperture head-on, but rather this relative response integrated over the rotation angle during which counts are being collected. For an ideal aperture with a squared off 'boxcar' relative response (see Figure 7), the aperture function is triangular shaped with half-power points at the same angles as the aperture edges. Due to geometric considerations, the actual relative response is a smooth quasi-Gaussian curve, and the aperture function is a similar smooth curve.

Using this aperture function and the known positions and strengths of the X-ray stars, count rates can be predicted (see Figure 8). Note that Cetus, Taurus, Scorpius, and the galactic center can be picked out easily, although in the case of IMP-J the galactic center appears as a lump on the side of the peak due to Scorpius. Knowing this information, one can generate dummy data for a specified date, given the position of the sun (which is tabulated in published ephermerides).

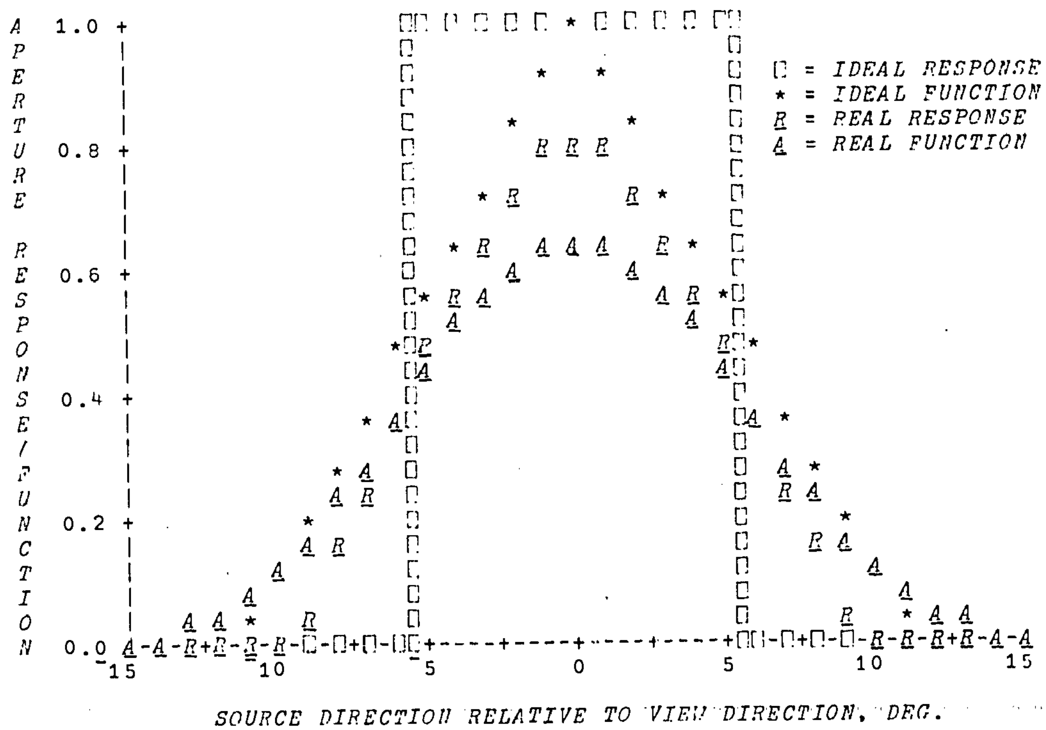


Figure 7. Aperture Functions.

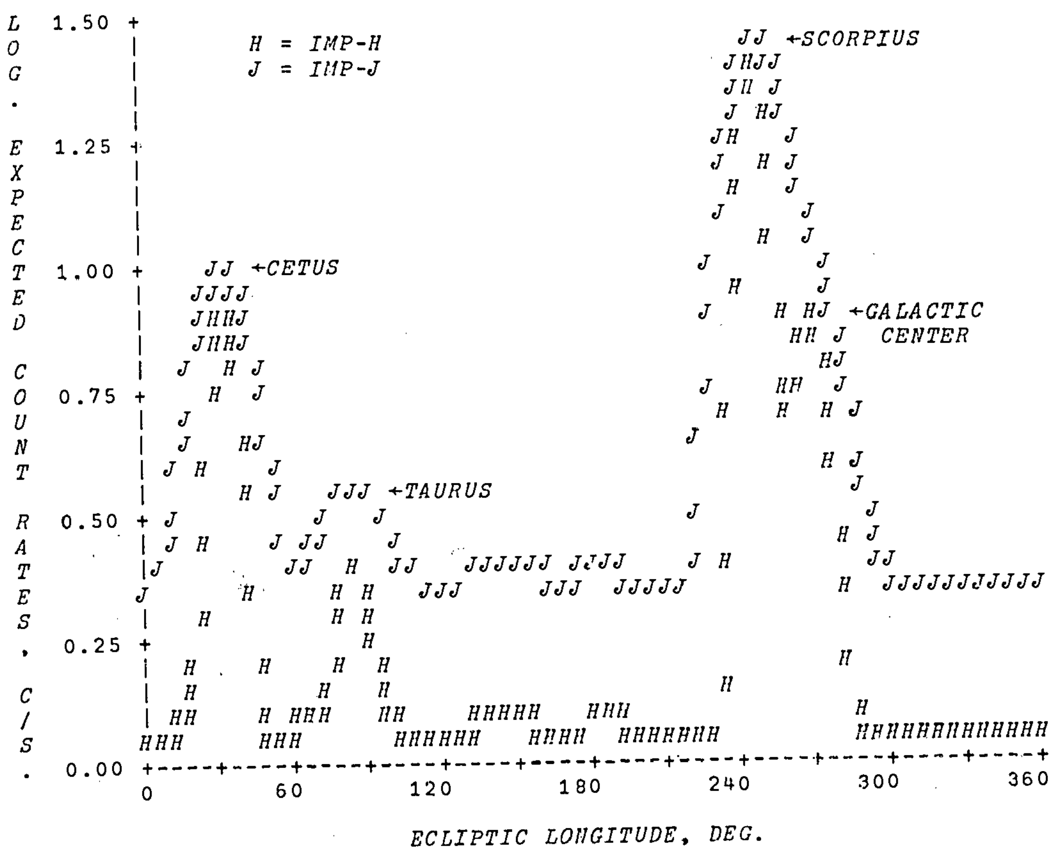


Figure 8. Expected Count Rates Versus Ecliptic Longitude.

In order to store these data compactly with the 36 Kbyte workspace, it was decided to convert them into binary words 18 bits long. The routine to do this conversion used the encode function followed by an '=1'. The rank was changed to make the data readable, although this change restricts the input data to ranks less than three. Note that the encode function actually increases core requirements until the '=1' operation is performed. Also note that the IBM XM6 version of APL is inconsistent in its binary encode-decode operations. For example, a 2 2 2 encode −1 results in a 1 1 1, while a 2 2 2 decode 1 1 1 results in a 7, not −1. The 'BIT' and 'BIT' routines given in the appendix use the first (leftmost) bit as a sign bit and correct for this error.

The piece de resistance of this effort comes, however, with the unfolding of the individual X-ray star count rates from data in which many of the stars are smeared together. The technique used derived from Chapter 9 of 'Data Reduction and Error Analysis for the Physical Sciences,' by P. R. Bevington, McGraw-Hill, 1969 (available in paperback). This technique employs a least squares-multiple regression analysis. Unlike the domino operator used dyadically, this approach determines, for an overdetermined set of equations, not only the coefficients but also a background coefficient and the uncertainties associated with all of these coefficients. Further, it allows the use of unequally weighted data.

The system of equations to be solved are shown on Figure 9. The Y's are measured with the uncertainties U, and the X's are known. The X's may be anything - polynomials, trigonometric functions, independent variables, etc. The coefficients A are to be determined with their associated uncertainties.

In this application, each sector count rate is a Y(i), and its uncertainty is U(i). Because of the low duty cycle due to limited telemetry and too few accumulators, 12 and 24 hour count rate sums will be used to determine the Y's. Poisson statistics apply, and the uncertainties are simply the square roots of the Y's. The accumulated counts and their uncertainties are scaled to a per second basis by dividing by the accumulation time (in seconds).

The sector count rates are equal to the background count rate (due to the diffuse component and charged particles) and the sum of the strengths of the individual X-ray stars, the A(j)'s, weighted by the aperture functions, the X(i;j)'s. The X(i;j)'s are functions of the star longitudes and the directions of view of the sectors. The view directions are known from the ecliptic longitude of the sun, and the X-ray star longitudes are known from star catalogs or may be derived from data taken over a period of time (e.g., Figure 8, on which many star longitudes may be found). Unless a star was observed within a sector, the corresponding X(i;j) will be zero. Rows of zeroes may exist, and these can be eliminated by using the compression operator.

One then defines the correlation matrix, or correlation coefficient matrix, RJK, in terms of the sample covariance matrix SJK, and the sample variance vector SJ. RJY is the linear correlation vector between the Jth variable, X(;j), and the dependent variable Y. The effect of having unequally weighted data is carried through by the constant 'C' and by the 1/U*2 terms.

In determining the correlation matrix RJK and the correaltion vector RJY, two obvious problems can occur; SJ or SY may have terms which are zero, thus causing domain errors; or RJK may be singular, which causes problems as it must be inverted. In the program 'SIEQ' (for simultaneous equations) these conditions are tested for, the first through a 0 or-dot-equal SJ, SY statement, and the second through the use of any handy determinant routine. If either SY or SJ has a zero term, or if the determinant of RJK is less than some arbitrary value, such as 10*-15, the program will branch to a step which pragmatically throws out one of the original equations, and then will branch to the beginning of the program. Since we are dealing with an overdetermined set of equations, the effect of throwing out an equation is to reduce the degrees of freedom by one, perhaps increasing the errors slightly, and to permit the program to run without abandoning the user to a domain error. Of course, the program also checks to ascertain that the set of equations is still overdetermined. If the set is no longer overdetermined, the user is flagged and a result of iota 0 is returned. The coefficients resulting from this analysis are given on Figure 10. The program 'SIEQ' returns these data as an 1; by 2 matrix.

The proof of the pudding, naturally, lies in the eating. Dummy data were used next in an analysis routine which also needed, as input, a vector of the longitudes of the X-ray stars. The unfolded data resulting from this routine should be a set of constant count rates and their corresponding uncertainties.

For the first trial, a small set of X-ray star longitudes was used - Scorpius, Taurus, Cetus, and the galactic center. The results are shown on Figure 11, hardly an example of smooth unfolding! Next, a source catalog of seven longitudes was used. The results are shown on Figure 12. Note the relatively straight lines. The variation in Scorpius is typewriter digitization noise, and the variation in the Galactic center is due to the fact that the galactic center is a clump of many X-ray sources. Clearly in the first trial (four sources) too few stars were used, and the program had to vary the fluxes to obtain the best fit. In the second case (seven sources), there were enough sources for the program to provide a fairly good fit. The numerical

```
TO SOLVE FOR THE A'S IN THE SET OF EQUATIONS:

     Y[1],U[1] ← A[0] + (A[1]×X[1;1]) + ... + (A[J]×X[1;J]) + ... + A[N]×X[1;N]
     Y[2],U[2] ← A[0] + (A[1]×X[2;1]) + ... + (A[J]×X[2;J]) + ... + A[N]×X[2;N]
     .    .    .    .    .            .    .            .    .
     .    .    .    .    .            .    .            .    .
     Y[I],U[I] ← A[0] + (A[1]×X[I;1]) + ... + (A[J]×X[I;J]) + ... + A[N]×X[J;N]
     ( THE U'S ARE THE UNCERTAINTIES IN THE MEASURED Y'S, X'S ARE KNOWN)

DEFINE:

     COVARIANCE MATRIX:   (SJK[J;K]*2) ← C×+/(÷U*2)×(X[;J]-X[J])×(X[;K]-X[K])
     COVARIANCE VECTOR:   (SJY[J]*2)   ← C×+/(÷U*2)×(X[;J]-X[J])×(Y-Y)
     SAMPLE VARIANCE:     (SJ[J]*2)    ← (SJK[J;J]*2)
     STANDARD DEVIATION:  (SY*2)       ← C×+/(÷U*2)×(Y-Y)*2
     CORRELATION MATRIX:  RJK[J;K]     ← (SJK[J;K]*2)÷SJ[J]×SJ[K]
     LINEAR CORREL. VECTOR: RJY[J]     ← (SJY[J]*2)÷SJ[J]×SY

WHERE:

     X[J] ← (+/(÷U*2)×X[;J])÷+/U*2
     Y    ← (+/(÷U*2)×Y)÷+/U*2
     C    ← (N÷N-1)÷+/U*2
```

Figure 9. Definition of the Multiple Linear-Regression Problem.

```
RESULTS:

     A[J] ← (SY÷SJ[J])×(RJY+.×⍉RJK←⌹RJK)[J]     J ← 1,2,...,N
     A[0] ← (÷+/U*2)×+/(÷U*2)×Y-X+.×A[1,2,...,N]
     (UA[J]*2) ← C×RJK[J;J]÷SJ[J]*2
     (UA[0]*2) ← C×((N-1)÷N)+(÷N-1)×+/((X[J]*2)×RJK[J;J]÷SJ[J]*2)+
                     +/X[J]×X[K]×RJK[J;K]÷SJ[J]×SJ[K]
     CHI-SQUARED ← +/(÷U*2)×(Y-X+.×A[1,...,N])*2
     DEGREES OF FREEDOM ← I-N-1
```

Figure 10. Solution of the Multiple Linear-Regression Problem.

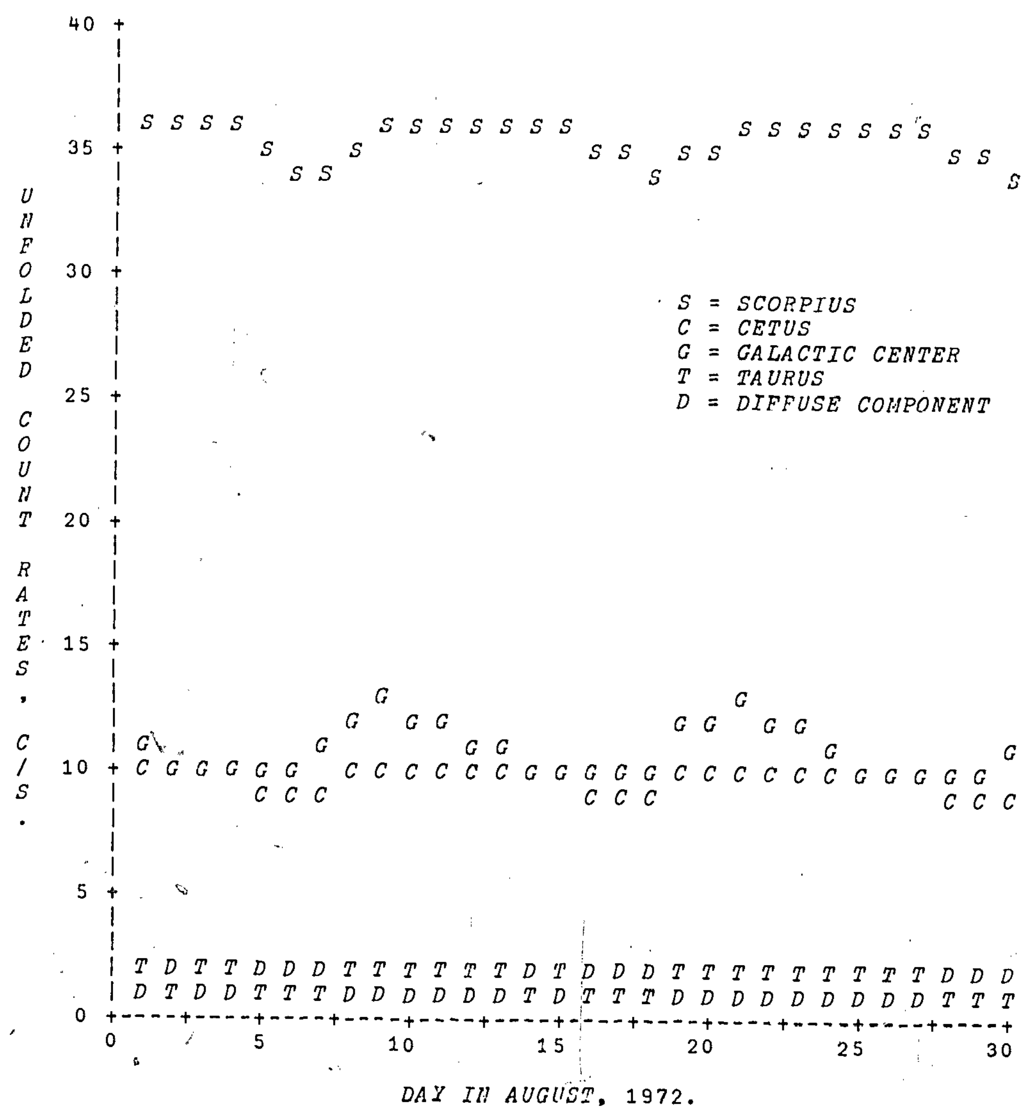


Figure 11. Unfolded Data Using Four Sources.

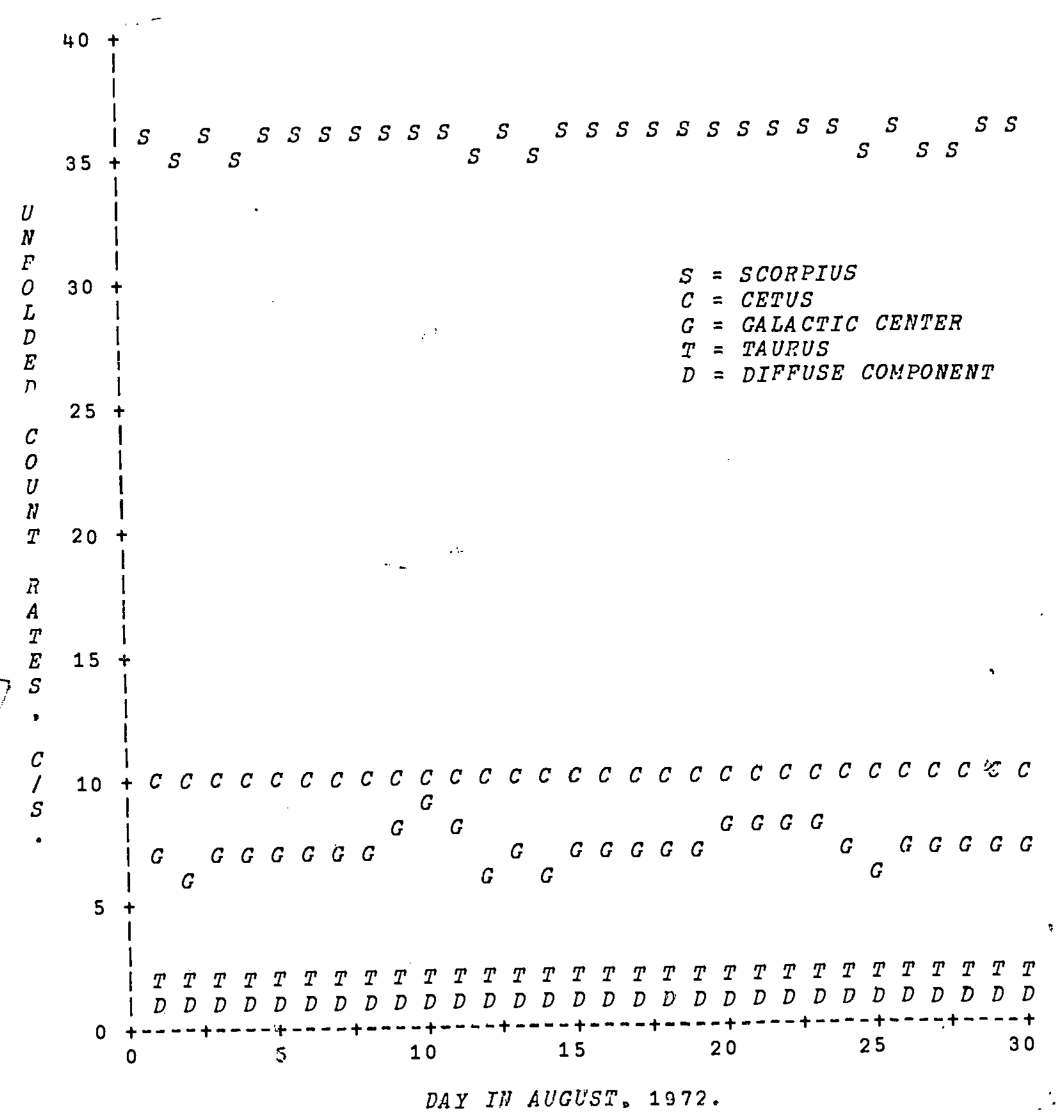


Figure 12. Unfolded Data Using Seven Sources.

DATA ACCUMULATED OVER 1 INTERVAL.

SOL. LONG.							
START:	129.30	130.30	131.20	132.20	133.10	134.10	135.10
STOP:	129.30	130.30	131.20	132.20	133.10	134.10	135.10
T[ACC]:	171.40	171.40	171.40	171.40	171.40	171.40	171.40
SOURCES:							
BKGND:	1.18	1.19	1.19	1.20	1.18	1.17	1.16
	0.07	0.06	0.07	0.07	0.11	0.12	0.12
SUN:	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0
	6.1	6.1	6.1	6.1	6.1	6.1	6.1
25.7	0.97	0.87	0.91	0.98	0.95	0.93	0.90
	0.35	0.37	0.36	0.34	0.35	0.33	0.31
32.9	9.71	9.78	9.73	9.68	9.74	9.79	9.82
	0.42	0.42	0.40	0.38	0.37	0.38	0.39
83.4	2.13	2.11	2.09	2.10	2.13	2.17	2.18
	0.21	0.21	0.22	0.23	0.28	0.31	0.33
245.2	35.60	35.37	35.57	35.45	35.59	35.59	35.53
	0.70	0.65	0.62	0.60	0.61	0.61	0.60
258.2	2.63	3.04	2.61	2.71	2.50	2.54	2.50
	1.08	0.78	0.60	0.47	0.41	0.38	0.40
266.0	6.67	6.35	6.73	6.79	6.97	7.04	7.17
	0.99	0.64	0.50	0.44	0.44	0.48	0.58
273.0	4.60	4.66	4.45	4.43	4.33	4.36	4.36
	0.46	0.36	0.34	0.35	0.40	0.47	0.55

Figure 13. Tabular Unfolded Data Using Seven Sources.

results are tabulated on Figure 13. Note that the variation in the flures due to the unfolding algorithm is usually less than 30 per cent of the uncertainty due to the count rate statistics.

The above analysis scheme was used on dummy data from both IMP-H and IMP-J to determine the feasibility of changing IMP-J from a 32 sector system to a 16 sector system. It was shown to be feasible, and this change will increase the duty cycle by a factor of 4, thus increasing the statistical precision by a factor of 2.

The utility of this study does not end with the launch of the satellites. Recently an extended file system was added to the Goddard APL system. This addition allows data tapes from the experiment to be placed on a disc where they can be accessed by the APL system. With only minor changes, these same analysis routies will be used to unfold the satellite data. The same programs used to optimize potential data prior to launch will then be used to analyze the actual data after launch.

ACKNOWLEDGEMENTS

Dr. S. M. Krimigis and his colleagues at the Applied Physics Laboratory of the Johns Hopkins University are to be thanked for inviting the author to participate on their IMP-H/IMP-J experimenter team. Thanks are also due to Dr. J. I. Vette and L. R. Davis for their help and encouragement and for making the facilities available for doing this work.

APPENDIX. PROGRAMS.

```
    ∇ Z←N BIT Y
[1]   ⍝ 'BIT' CONVERTS DATA Y (3>⍴⍴Y) INTO BINARY
[2]   ⍝ WORDS N BITS LONG (INCLUDING SIGN BIT).
[3]   Z←(Y<0),[1] 1=((N-1)⍴2)⊤⌊0.5+(¯1+2*N-1)⌊(¯2*N-1)⌈Y
[4]   Z←((1⌈(1↑⍴Y)×1<⍴⍴Y),(N×1⌈¯1↑⍴Y))⍴,⍉(N,1⌈×/⍴Y)⍴,Z
    ∇
```

```
    ∇ Z←N BIT Y
[1]   ⍝ 'BIT' UNCONVERTS BINARY DATA CONVERTED TO BINARY
[2]   ⍝ FORM BY 'BIT'.
[3]   Z←(N,(¯1↓⍴Y),((¯1↑⍴Y)÷N))⍴,⍉Z←(((×/⍴Y)÷N),N)⍴,Y
[4]   Z←((1↓⍴Z)⍴,-((1,1↓⍴Z)↑Z)×2*N-1)+2⊥(1,(¯1+⍴⍴Z)⍴0)↓Z
    ∇
```

```
    ∇ Z←N SHO X
[1]   ⍝ 'SHO' DISPLAYS BINARY DATA GENERATED BY 'BIT' IN AN
[2]   ⍝ EASILY READABLE FORM.
[3]   Z←((((¯1↑⍴X)÷N)×1+N)⍴(N⍴1),0)\'01'[1+X]
    ∇

      ⎕←X←4 5⍴¯11+⍳20
¯10 ¯9 ¯8 ¯7 ¯6
 ¯5 ¯4 ¯3 ¯2 ¯1
  0  1  2  3  4
  5  6  7  8  9
```



```
    ∇ Z←Y SIEQ X;X;Y;X̲;Y̲;C
[1]   Z←⍳0
[2]   →(>/⍴X)/GO
[3]   ¯1↑⍴X;' UNKNOWNS, ';1↑⍴X;' EQUATIONS.'
[4]   →0
[5] GO:C←(⍴SSI)÷(¯1+⍴SSI)×+/SSI←÷Y[;2]*2
[6]   RJK←X-((1↑⍴X)⍴0)∘.+X̲←(SSI+.×X)÷+/SSI
[7]   SY←(C×SSI+.×(Y[;1]-Y̲←(SSI+.×Y[;1])÷+/SSI)*2)*
      0.5
[8]   →(0∨.=SY,SJ←(C×SSI+.×RJK*2)*0.5)/CHP
[9]   RJY←(C×(SSI×Y[;1]-Y̲)+.×RJK)÷SY×SJ
[10]  RJK←(C×(⍉(SSI∘.+(⍴X[1;])⍴0)×RJK)+.×RJK)÷SJ∘.×SJ
[11]  →((1E¯15)<DET RJK)/OK
[12]  ⍝ 'DET' IS ANY HANDY DETERMINANT ROUTINE, SUCH AS
[13]  ⍝ 'D←DET M' ( ⎕,III,NO.4,P.33,FEB. 10,1972), I.E.,
[14]  ⍝ →×⍴,M[×⍴M;]←D×M[×⍴M←1 1↓M-M[;1]∘.×M[1;]÷D←1⍴M;]
[15] CHP:X←(Z←(¯1*2|1↑⍴X),0)↓X
[16]  Y←Z↓Y
[17]  'DELETE 1 FROM THE ',(7×1↑Z)↑'BOTTOM.TOP.
[18]  →1
[19] OK:Z←SY×(RJY+.×⍉R̲J̲K̲←⌹RJK)÷SJ
[20]  Z←((÷+/SSI)×+/SSI×Y[;1]-X+.×Z),Z
[21]  Z←Z,((÷+/SSI)+C×+/((X̲*2)×(÷ 1 1 ⍉SJ)× 1 1 ⍉R̲J̲K̲)++/(X̲∘.×X̲)×
      R̲J̲K̲÷SJ+SJ∘.×SJ)*0.5
[22]  Z←⍉(2,0.5×⍴Z)⍴Z←Z,((C÷ 1 1 ⍉SJ)× 1 1 ⍉R̲J̲K̲)*
      0.5
[23]  CHISQ←(+/SSI×(Y[;1]-X+.×1↓Z[1;])*2),((-/⍴X)-1)
[24]  X←Y←C←X̲←Y̲←⍳0
    ∇
```

```
      ⎕←X←4 BIT X
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0
1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1
0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0
0 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1
```

```
      4 SHO X
1000 1000 1000 1001 1010
1011 1100 1101 1110 1111
0000 0001 0010 0011 0100
0101 0110 0111 0111 0111
```

```
      4 BIT X
¯8      ¯8      ¯8      ¯7      ¯6
¯5      ¯4      ¯3      ¯2      ¯1
 0       1       2       3       4
 5       6       7       7       7
```

REPRESENTING NEGATIVE INTEGERS IN BIT VECTORS

A Short Note

L. J. Woodrum
IBM Corporation
Poughkeepsie, N. Y. 12601

To represent a positive or negative integer as a bit vector, "two's complement" arithmetic may be used. By using a radix vector, *R*←2 ¯2,(*D*-2)⍴2, if *N* is a positive or a negative integer, and (*N*≥-2⋆*D*)∧*N*<2⋆*D* is true, then *X*←*R*⊤*N* makes *X* a bit vector in two's-complement form. Similarly, *R*⊥*X* is the scalar *N*.

APL AS A TEACHING TOOL: TWO VERSATILE TUTORIAL APPROACHES

Leslie N. Davis, Jak Eskinazi and Daniel J. Macero
Syracuse University
Syracuse, New York 13210

Introduction

Computer assisted instruction has played a significant role in several undergraduate courses in the Chemistry Department at Syracuse University. Programs for drill, tutorial[1,2,3], and simulated laboratory procedures have been implemented for an introductory course for non-science majors, while programs for data analysis, tutorial instruction[4], and examinations are used in an upper level course for chemistry majors. Our experience leads us to believe that interest in and implementation of the computer as a classroom tool will continue to grow here and elsewhere, and programs using the tutorial approach will contribute significantly to this growth.

By "tutorial" we mean that the program gives the computer the capability to adjust to each student's needs on an individual basis not only with respect to the depth and speed of coverage of the material presented, but also for the analysis of specific student errors when they occur, be they mechanical in nature (i.e., dividing by 10 instead of 100) or conceptual (i.e., trying to work pH problems before understanding logarithms). Because of the extraordinary versatility of the APL language, there are as many ways to approach tutorial program design as there are teaching personalities. This paper describes two approaches that we have taken. Much of their significance is their non-limiting open-ended design which suggests applications outside the field of chemistry or even science.

Our APL programs are accessed on the University's IBM 370/155 computer via more than one hundred IBM 2741 terminals located all over the campus, including five in the chemistry building.

Program Design

The basic decision in the development of any APL-CAI program is the kind of response the student is expected to input to the computer. The programs we have chosen to discuss, while fundamentally similar in their tutorial design philosophy, are distinct in their methods of inputting student responses.

The pH-logarithm program deals with subject matter and student responses exclusively numerical in nature. This type of material allows an essentially infinite number of problems to be randomly generated by the computer. It is noteworthy that the actual student input is accepted by this program as APL literal, not numeric data, because the analysis of significant figures, juxtaposed digits, etc., is possible only on literal input.

The programs in chemical instrumentation require sentences or phrases for most responses. In order to fulfill this requirement and the additional one that data outputting for these programs be mutually compatible, the multiple choice format is utilized for student responses. This also provided needed flexibility in individual program design and allowed much larger question libraries for the available workspace size than would have been possible using other input systems.

Self-Teaching pH and Logarithms

When introducing undergraduates, particularly at the freshman level, to the concept of pH it is often found necessary to reinforce and improve their background in logarithms. Most students at this level will not otherwise be able to use logarithms efficiently for simple conversions between pH and hydrogen ion concentration. To save valuable classroom time and be able to bring all students up to the same level of proficiency at their own pace, we decided to develop an interactive APL program.

Although student responses are numerical, input is accepted as a character string. This allows an incorrect answer to be literally dissected in order to give a clue as to where the error lies. Even when a control word is entered a check is made to see if it is a valid word, if the proper number of characters have been entered, if two characters have been transposed, or if incorrect characters (up to two) have been entered.

The CAI program is divided into sixteen units (Table I). The computer adjusts itself to the student's rate of progress by requiring two successive correct answers before moving on to the next unit. Within a given unit the basic format of the question is the same; however, since each

TABLE I

TOPICS COVERED IN pH AND LOGARITHMS PROGRAM

UNIT	DESCRIPTION
A	Review of exponents
B	Logarithms to different bases
C	Log of powers of ten
D	How to use tables to find logs
E	Antilogs of simple numbers
F	Using logs in multiplication and division
G	Log of numbers greater than ten
H	Antilogs of positive numbers
I	Log of numbers smaller than one
J	Antilogs of negative numbers
K	Using logs in multiplication
L	Using logs in division
M	Finding powers with logs
N	Taking roots with logs
O	A chemical application, finding pH
P	Finding [H+] from pH
Q	Interpolations in finding logarithms
R	Interpolations and antilogs

TABLE II

CONTROL WORDS FOR LOG PROGRAM

WORD	DESCRIPTION
CALCULATION	Allows use of computer as desk calculator.
EXAMPLES	Gives answer to present question and one example on the current material.
INFORMATION	Depending on question, either solves a similar problem in step-by-step detail or gives a hint on how to solve it.
COMMENT	Allows student to enter comments at any time on any aspect of the program.
SKIP, X	Skips ahead to unit designated by X and puts control in hands of student (i.e., answers are not evaluated by computer) until control word '*CONTINUE*' is entered.
REVIEW, X	Reviews unit designated by X. Otherwise same as '*SKIP*'.
REPEAT	Causes repetition of previous question until '*SKIP*', '*REVIEW*', or '*CONTINUE*' command is used.
CONTINUE	Skips to next consecutive unit with computer taking control of the program.
STOP	Stops program, gives sign-off information.

number in an example is randomly generated, each question is different regardless of how many times it is repeated.

At the start of each unit a typical problem from the unit is solved for the student without going into much detail. Then a different problem of similar difficulty is presented to the student for him to solve. A student who is using the program as a review exercise should be able to solve the problem and move on without further delay. However, a student who has no idea how to begin to answer the question will require further help. He can get help by entering the word INFORMATION as a response to the question. This control word directs the computer to solve a problem of similar difficulty in step-by-step detail. Since each number is randomly generated, INFORMATION can be requested as many times as the student desires. After the execution of INFORMATION the computer restates the original problem and awaits the student's response.

The student has nine control words at his disposal (Table II). SKIP enables him to skip material he feels competent with and allows more time for new material. If he feels the need for more detail in a given segment, REVIEW allows him to back track to another unit. STOP suspends the program and gives sign-off information. At the next sign-in the program resumes exactly where it left off when the word STOP was entered. The other control words are explained in Table II.

The primary aim of this program is to introduce the student to the concept of pH and develop sufficient skill with logarithms for satisfactory pH manipulations. The following typical student-computer interaction illustrates the method.

First, the subject is presented and an example is given:

O) <u>THE CONCEPT OF PH</u>

IF THE HYDROGEN ION CONCENTRATION ([H+]) *IS* 4.63E¯5 *MOLES/LITER,*

THE PH OF THE SOLUTION IS 4.3344.

Then a question is asked for the student to solve:

O) GIVEN THAT THE HYDROGEN ION CONCENTRATION IS 3.24E¯4

THE PH OF THE SOLUTION IS ______________

after typing the lines and backspacing to the first underline, the computer waits for the student entry.* He has the option of entering his answer or a control word. Suppose he enters INFORMATION. The computer responds:

<u>THE CONCEPT OF PH</u>

WHEN THE CONCENTRATION OF HYDROGEN ION IN SOLUTION IS EXPRESSED AS GRAM-IONIC WEIGHTS PER LITER, THE PH (LOWER CASE P, UPPERCASE H) OF THE SOLUTION IS DEFINED AS THE <u>NEGATIVE</u> LOGARITHM OF THIS CONCENTRATION.

FOR EXAMPLE, WATER AT 25 *DEGREES CENTIGRADE HAS A HYDROGEN ION CONCENTRATION [H+] OF* 1E¯7 *AND OF HYDROXIDE ION [OH¯] OF* 1E¯7 *MOLES/LITER. THUS THE PH OF WATER IS* 7 *AT* 25 *DEGREES.*

TO SUMMARIZE, TO FIND THE PH TAKE THE LOG OF THE HYDROGEN ION CONCENTRATION AND CHANGE ITS SIGN.

IF YOU DO NOT KNOW HOW TO FIND THE LOG OF NUMBERS SMALLER THAN ONE, REVIEW UNIT I.

---

*The quad-quote input-output used in this program is unique to the SU Computing System, but could be incorporated into any APL system.

Now the question asked above is repeated. A student who wished to review Unit I would enter REVIEW, I. Let us assume the student responds with an answer, 3.8495, instead of the correct answer 3.4895 (*i.e.*, second and third digits transposed). The computer responds as follows:

JOHN, TWO OF YOUR NUMBERS ARE WRONG.

THE PH OF THE SOLUTION IS 3.??95

The question is now restated. If the correct answer is entered, the computer responds:

VERY GOOD! GIVEN THAT THE HYDROGEN ION CONCENTRATION IS 3.24E-4

THE PH OF THE SOLUTION IS 3.4895

The student is given credit fr the answer and further reinforced by seeing the correct answer printed out once more. At this time, depending on the circumstances preceding the question, the computer will either ask another question in the unit or move to the next unit.

In view of the fact that this program is designed to instruct and drill students, not test their knowledge of the material, we did not see the need to gather data on their progress. If needed, however, all data pertinent to the use of the program can be coded and stored for later retrieval without altering the performance of the system in any way. It is also possible to disable the control words and to transform the program into a type of examination which will ask sixteen questions (one from each unit) and record student answers as either right or wrong. This would require minimal editing of the subprograms. The flexibility of the program is further reflected in the fact that questions may be added or deleted at any time providing the same skeletal framework of the program is maintained.

CAI Topics in Chemical Instrumentation

The interactive programs used in our upper level course in chemical instrumentation allow the student to learn, self-test, and review several specialized topics at a rate of progress adjusted to the student's responses. In programming a given topic, the subject matter is divided into four to six groups of questions which are further subdivided into three of four subgroups of related questions. The method of question presentation is similar to that described by Castleberry and Lagowski[5]. The student is first presented with a question from Group I, Subgroup 1. If he answers it correctly the program chooses the next question randomly from Group I, Subgroup 2, then from Subgroup 3, and then from Group II, Subgroup 1, etc. Student mistakes result in hints which may help lead him to the correct answer. The hints try to offer a brief discussion to clarify the student's thinking. A mistake followed by a correct answer causes the next question to be selected from the same subgroup in which the mistake was made to make sure the concept under consideration is well understood and to check that the answer was not guessed at. A typical terminal printout from the ELECTRONICS program follows, including both student entries and computer responses:

```
                                  Z2
                              ----|(----
                              |         |
                              |   |\    |
1 VOLT PP o--/\/\/\---|---|-\   |
SQUAREWAVE      Z1      S     |   \ |  o
                                  |   >- --
                                  |  /
                              T--|+/
                              |   |/
                              V
```

WHEN E(IN) IS IN THE POSITIVE HALF OF THE CYCLE, THE VOLTAGE AT S IS A. > E(IN)   B. > E(OUT)   C. < E(OUT)   D. E(OUT) - E(IN)

C

SINCE E(IN) IS +, E(OUT) MUST BE -. WHERE DOES THAT LEAVE S?

B

CORRECT! E(IN) AND E(OUT) ARE OF OPPOSITE POLARITY. IF E(OUT) IS - IT IS AT A POTENTIAL LOWER THAN GROUND.

WHEN E(IN) IS IN THE POSITIVE HALF OF THE CYCLE, THE POTENTIAL ACROSS Z2 IS A. <0   B. 0   C. >0   D. ALWAYS = E(IN)

A

RIGHT YOU ARE, JOHN! SINCE E(OUT) IS NEGATIVE, POTENTIAL ACROSS Z2 < 0.

YOU ARE DOING WELL, JOHN! KEEP IT UP!

The progress of the student is monitored continuously and the computer will end the session if the student is scoring less than 80% correct answers, subject to certain other conditions. If a student is signed-off by the computer he is advised where his weaknesses are and is invited to try again after further study and consultation with his instructor.

The data generated by the students as they use the programs is coded and stored in one literal and two numeric vectors which are automatically extended each time a program is used in a given workspace. Data from student workspaces is copied periodically by the instructor into a central data managing workspace where output programs store it and/or print it out in tabular form. Data from any of the programs can be stored and accessed intermixed in any order.

Upon successful completion of an entire program, the computer prints out a special certificate (Figure 1) which attests to the student's having achieved a certain level of competence in that particular subject area. Few students ask about a numerical "grade" once they have finished a program. Since they are unaware of the exact criteria on which they are getting through they tend to pay more attenton to the content of the programs and less on keeping track of the number of right and wrong answers.

Since the programs are skeletally identical irrespective of subject matter, editing individual questions is a simple affair, requiring minimal APL experience.

Table III lists the program titles used in the instrumentation course and the topics covered.

Student reactions to these programs have been very favorable. Most of our upper level students are far more anxious to try CAI than the typical freshman. Once they realize they are not being graded by the computer they settle down to the challenge of getting it to print out their certificate as soon as possible. The programs have provision for students to enter comments about them; about 30% of the students take advantage of this opportunity, the majority of these expressing positive attitudes such as, "I found it very interesting and most important of all, it is a good way of learning", and "I wish we had things like this in freshman and organic chemistry", and, "It was fun." Occasionally a student will mention how or why a particular question confused him and this has led to periodic revision of the question libraries.

Students who have used the CAI programs have done better on midterm and final written examinations than students who were not exposed to the programs. The written examinations given are generally of the problem-solving type and never repeat questions in the computer libraries. Castleberry and Lagowski[5] have encountered a similar effect of CAI on exam grades in a freshman chemistry course.

Conclusions

Tutorial teaching programs can play an important part in chemistry courses on any level. Regardless of the nature of the material or the type of desired student response, it is usually possible to design tutorial programs which are both interesting and useful to the students. Our experience has been that these programs free the instructor to do more individualized teaching, assure a measured minimum level of competence on the part of the students, and most important, allow this competence to be achieved at a rate determined mainly by the students' interests and abilities. It is our intention to continue to expand the scope of the courses we are involved in with additional CAI materials.

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

6/19/72

T O   W H O M   I T   M A Y   C O N C E R N :

PLEASE BE ADVISED THAT ON THIS DATE,

```
→→→→→→→→→→→→→→→                          ←←←←←←←←←←←←←←←
→→→→→→→→→→→→→→→→→                      ←←←←←←←←←←←←←←←←←
                       BILL BROWN
→→→→→→→→→→→→→→→→→                      ←←←←←←←←←←←←←←←←←
→→→→→→→→→→→→→→→                          ←←←←←←←←←←←←←←←
```

HAVING LABORIOUSLY DEVOTED MUCH TIME AND ENERGY TO THE FASCINATING STUDY OF

G A S   C H R O M A T O G R A P H Y

HAS SUCCESSFULLY AND HONOURABLY COMPLETED A COMPUTER ASSISTED PROGRAM THEREIN IN PARTIAL FULFILLMENT OF THE REQUIREMENTS IN THE COURSE OF STUDY IN

<u>C H E M I C A L   I N S T R U M E N T A T I O N</u>

WHEREBY, WE HEREON AFFIX OUR SIGNATURES IN TESTIMONY AND WITNESS TO THIS OUTSTANDING EVENT.

```
        o                       ω~ıΓ↑Δnι∘
    o       o                   IBM 370-155\APL,
  o           o                 PROFESSOR OF CAI
  ∘SYR.UNIV.∘
  o           o
    o       o
        o
     *  *  *
    ***  ***
   ****  ****
  *****  *****
 ******  ******
*******  *******
********  ********
```

Figure 1

TABLE III

CAI INSTRUMENTATION PROGRAMS

Program	Description
ELECTRONICS	Series and parallel networks, properties of VTVM's and oscilloscopes, passive networks, feedback, operational amplifier properties, simple operational amplifier circuits.
GAS CHROMATOGRAPHY	Separation theory, thermal conductivity detector theory and circuitry, specific operation of the Carle instrument used in the course.
SPECTROSCOPY	Basic theory of UV, visible and IR spectroscopy and instrumentation, and the use of instruments available in the course: Bausch and Lomb Spectronic 20, Beckman DB, and Perlin-Elmer 237B.

BIBLIOGRAPHY

1. Coffey, C. E. and Macero, D. J., "Computer Assisted Instruction in an Introductory Chemistry Course", paper given at the Finger Lakes Regional Computing Organization (FLARCO) at Eisenhower College, Seneca Falls, New York, April 24, 1971.

2. Coffey, C. E. and Macero, D. J., "Computer Assisted Instruction in Chemistry", talk given at IBM Filed Systems Center, Syracuse, New York, October 20, 1971.

3. Eskinazi, J. and Macero, D. J., "An Interactive Program for Teaching pH", J. Chem. Educ., in press (1972).

4. Davis, L. N. and Macero, D. J., "Computer Assisted Instruction in a Chemical Instrumentation Course", J. Chem. Educ., in press (1972).

5. Castleberry, S. and Lagowski, J. J., J. Chem. Educ., 47, 91 (1970).

THE EVOLUTION OF AN INTERACTIVE CHEMISTRY LABORATORY PROGRAM

Thomas R. Dehner and Bruce E. Norcross
State University of New York at Binghamton
Binghamton, New York 13901

Introduction

During the past several years a program has been developed at SUNY-Binghamton for the introductory Chemistry laboratory which allows the student to test his laboratory results in a particular experiment against those results expected for his sample. This test is performed during the regularly scheduled laboratory period, by the student, at a terminal located adjacent to the laboratory, and permits a rapid decision to be made by the student as to whether or not he should repeat the experiment in order to obtain worthwhile data. This operation is accomplished by way of the APL program CHEMLAB1, which performs the appropriate calculations on the student's raw data. At the time the repeat-no repeat decision is made, the student does not have, or get, the results of the detailed calculations made by the computer. Should a repeat be necessary, the student has enough time to perform a duplicate experiment on his relatively simple apparatus before the end of the lab period. Should no repeat be necessary, as is the case 60-70% of the time, the student can dismantle his apparatus and proceed with the calculations at his leisure, assured that the data is capable of providing a reasonable answer.

After the student has understood the concepts of the experiment, and performed the appropriate calculations, he may enter his calculated data at the terminal, and receive a tabular output which tells him how well he did on the calculations and in the experiment. These interactions of the student and the program are shown in Figures 1 and 2. This output is turned in with the student lab report, and serves as a basis for student comments on the relative success or failure of his experimental work and calculations.

The development of the program, and its use and reception by students, teaching assistants, and faculty have been described before in a general, user-oriented, non-technical, approach [1]. During the evolution of the program, a number of APL programming features were developed to accommodate the particular types of student interaction desired or observed. It is the purpose of this paper to discuss in detail some of these features, and how they accomplish the desired ends.

Some Programming Features

We have tried two kinds of student name entry for CHEMLAB. In the more general program, a student enters his name on request, and it is then stored, exactly as written, for future use. In the second version, a name table is filed initially with names of students in a particular laboratory section. These students must then enter that workspace for their computer experience. Both versions are in use, and have particular advantages for different purposes. The first method makes the program generally available to anyone. The second version is more efficient for large class production use.

Two kinds of data entry are available for this program. Originally a step-by-step method was used in which each separate item was called for individually. Such entry required a question-answer sequence for each datum; each sequence requiring an irreducible amount of transmission-response time. Since it was important to reduce the time at the terminal for each student during the first entry into the program, which occurs during a laboratory period, an attempt to cut the number of separate entries per student was explored.

The approach used was to have the students enter the primary data in vector form. A clear difference in time required for the initial session was found only when the students prepared for the vector entry by organizing their raw data in the order desired before signing on the terminal. This aspect can be useful in encouraging students to set up a proper data table in their lab notebooks. Otherwise, the directed nature of the step-by-step entry was more efficient. Under ordinary circumstances, a student should be at the terminal for less than five minutes for each part of CHEMLAB. Two terminals available for the last hour of a three-hour lab accommodate one section of 24 students without excessive delays. Two terminals available for the last two hours of such a three-hour laboratory period accommodate most of the students in two 24-student laboratory sections.

Error messages designed to correct faulty entry seem to stimulate some programmers to excesses of cuteness or sarcasm. While a light touch with prose responses is often stimulating and encouraging to the students, it is easy to misjudge the appropriateness of a response from consideration of only one situation. For instance, an early draft error message received by one who entered a quantity requested to be in liters with a five-digit number (obviously milliliters instead) was faced with the message:

I SAID LITERS, DUMMY. TRY AGAIN.

This message might be an appropriate rap on the knuckles for the bright but careless student who was secure in his understanding of the experiment and the calculation involved. It most decidedly was not a proper response for the ordinary introductory student becoming acquainted with the computer for the first time. That part of the program now is:

```
[152] RETURN←RETURN,(ι26)+1

[153] '|6| MOLAR VOLUME OF OXYGEN (IN LITERS)'

[154] →((TEMP≥10)∧((TEMP←⎕≤50))/AROUND4

[155] 'MOLAR VOLUME IN LITERS, PLEASE!

[156] →RETURN[ρRETURN]

[157] AROUND4:DATA[I,SWITCH,14]←TEMP
```

It can be seen that if the test in step 154 is not met successfully, a more appropriate message is given (step 155), and the program returns to the question (step 153) for a repeat.

It is important to place checks for decimal points or proper dimensions in a program which involves numerical input, with responses which will permit a student to find an acceptable answer before bouncing him from the program, with or without an error signal. For instance, an early version of the CHEMLAB program had the following sequence:

```
[62] '|4| TEMPERATURE OF THE OXYGEN [CENTIGRADE]:'

[63] DATA[I;SWITCH;5]←⌊TEMP←⎕

[64] →(0.5>TEMP-DATA[I;SWITCH;5])/AROUND

[65] DATA[I;SWITCH;5]←DATA[I;SWITCH;5]+1

[66] AROUND:(ι26)+FUDGY

[67] RETURN←RETURN,(ι26)+1

[68] '|5| BAROMETRIC READING; UNCORRECTED AT'; DATA[I;SWITCH;5]; '∘CENTIGRADE'
```

In step 62, the student is requested to respond to a question concerning gas temperature. His numerical answer, indicated by the open box at the extreme right-hand end of step 63, is stored in a location called TEMP. This value is shorn of any fractional part (by the APL operator XXXX) and the resulting integer is stored in a data array location named DATA[I;SWITCH;5]. Step 64 checks to see if the difference between TEMP (which is an integer) and DATA[I;SWITCH;5] is less than 0.5. If so, the rounded value is properly stored, and the program is branched to that step labelled AROUND (step 66), which in turn sends the program to step 68 - a new question. However, if the value found in step 64 is <u>not</u> less than 0.5, the program does not branch to AROUND, but falls through to step 65, which increases the value stored in DATA[I;SWITCH;5] by 1. These four steps in effect round off the temperature values entered to the nearest degree (later versions of this program perform this operation more efficiently). This temperature value, stored in DATA[I;*SWITCH*;5], will be used later as an index to pick out a vapor pressure correction from a table stored in the workspace. This value of the vapor pressure of water at the temperature of the experiment is then used in a calculation of a computer-derived result which is used as the basis for comparison with the student-calculated result.

The programming problem and solution outlined above ignores two alternative possible student responses: the temperature measured may fall outside either extreme of the temperature table, or, the student may make a typing or understanding error in entering the datum so that the number entered is not a temperature value at all. Since this is the kind of error such a program should recognize, a checking sub-routine has been introduced:

```
[52] RETURN←RETURN,(ɪ26)+1

[53] '|4| TEMPERATURE OF THE OXYGEN [CENTIGRADE]:'

[54] →((TEMP≥16)∧((TEMP←⎕)≤33))/AROUND

[55] 'DATA OUTSIDE RANGE OF TEMPERATURE CORRECTION TABLE; 16≤T≤33∘C'

[56] →RETURN[⍴RETURN]

[57] AROUND:DATA[I;SWITCH;5]←TEMP

[58] →(ɪ26)+FUDGY

[59] RETURN←RETURN,(ɪ26)+1

[60) '|5| BAROMETRIC READING; UNCORRECTED AT ROOM TEMPERATURE.'
```

In this sequence, step 53 asks for the observed temperature. Step 54 tests the entered value, stored in TEMP, to see if it falls within the range of 16-33 degrees C. If this requirement is met, the program branches to the next "AROUND", which is step 57, and the value of TEMP is stored in position DATA[I;SWITCH;5].

Should this requirement not be met, the program proceeds to the next line, which is an error message:

DATA OUTSIDE RANGE OF TEMPERATURE CORRECTION TABLE; 16≤*T*≤33∘*C*

After this printout, the next step, 56, returns the program to step 52, which begins the question-and-answer sequence again. Checks of this type have been introduced at many points in the program.

Another major feature of this program is the inclusion of a routine which permits a student to examine his input data, identify a mistake, and correct it without having to reenter all data. An example of the steps which accomplish this are found in lines 72-82 of the CHEMLAB program, Figures 3 and 4. Since this question-retry routine is one of the more opaque segments of this program, a detailed analysis of the steps follows (some earlier steps are included in the analysis, since they set indicators necessary for the branching routines):

Line 37: This line is the beginning of Part 1.

Line 39: FUDGY is an index to control branching. It is initially set to 1 so that the operation →(ɪ26)+*FUDGY* will yield a branch to the succeeding line. Later on, FUDGY will receive values which will cause branching to predetermined locations within the function.

Line 40: RETURN is a vector composed of the statement numbers associated with data input statements. It is constructed by catenating together the statement numbers of these lines as the student makes his first pass through the input statements. Should the student require an updating of information already entered, a branch can be made back to the appropriate line by way of these statement numbers. A statement of this kind is used before every data entry.

Line 43: Line 43 is a variable branch statement. When FUDGY has a value of 1, the branch is to the following line. When FUDGY has some other value (calculated in line 81), the branch is to that statement specified by ɪ26 (the statement being executed) plus the value of FUDGY. This statement appears after each data entry.

Line 77: RETURN is the vector of statement numbers; its last element is the statement number associated with the beginning of the Question Retry Area. GOTO is a vector composed of the statement numbers associated with only those areas the student wishes to retry. GOTO's last element is the same as RETURN's last element.

Line 78: GATE is a matrix composed of question entry points matched with corresponding exit points. Example: Line 66 is the entry point for Question 3, and Line 68 is its exit point.

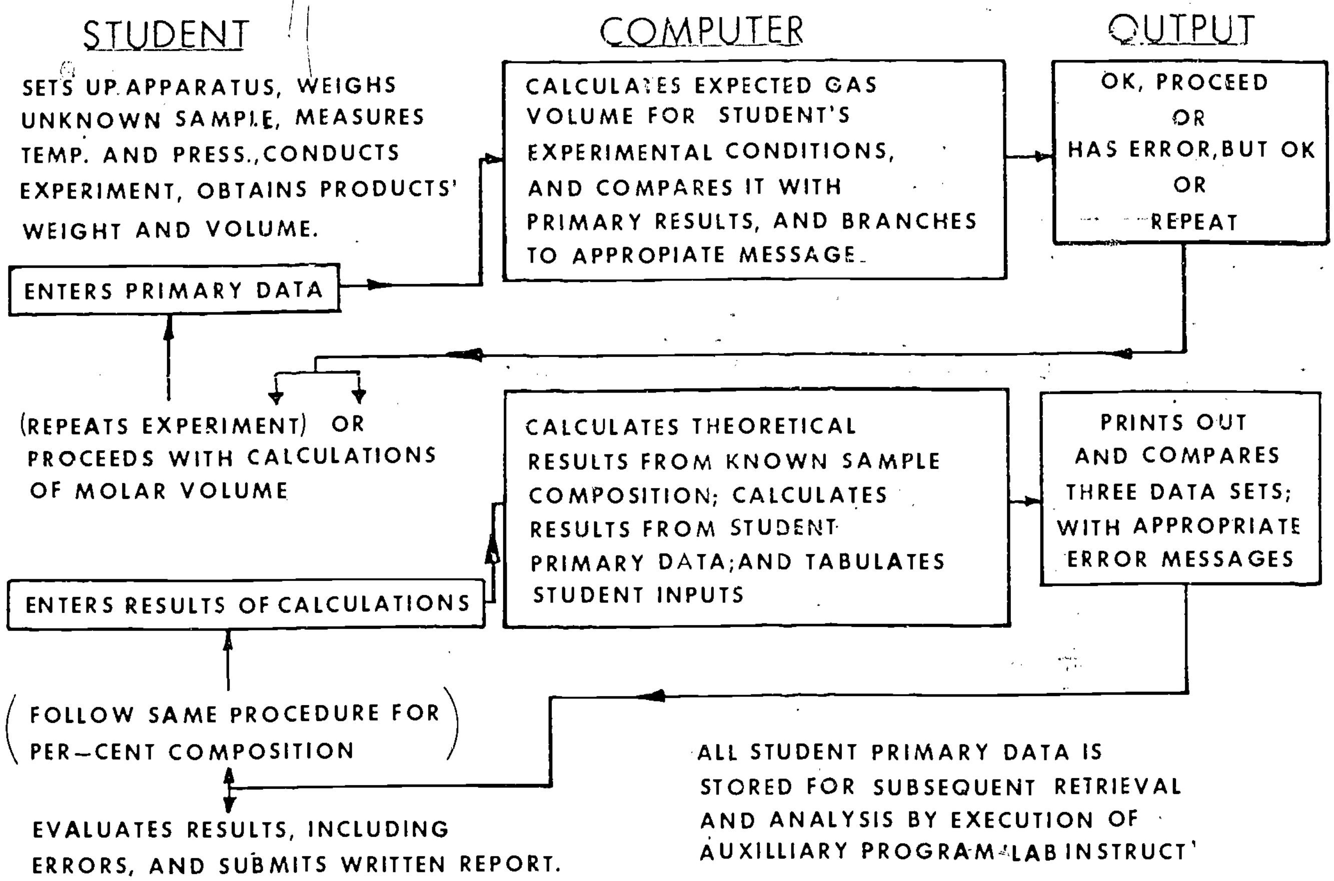


Figure 1. Student Interaction with *CHEMLAB*1

```
                                                 |--------MACHINE COMPUTED
                                                 |        USING YOUR DATA.
                                 YOUR            |
                                 DATA.           |        MACHINE COMPUTED
                                                 |     |-- THEORETICAL
                                   |             |     |    VALUES.
                                   ↓             ↓     ↓

WEIGHT OF OXYGEN.                 0.3186*        0.3186   0.3437
ABSOLUTE TEMPERATURE.           297.0          297.2    297.2
CORRECTED BAROMETRIC
    READING.                    739.3          739.1    739.1
PRESSURE DUE TO OXYGEN.         716.9          716.7    716.7
VOLUME OF O2 AT STP.            219*           217      241
MOLAR VOLUME OF O2.              22.2           21.8     22.4

  COMPARISON OF THESE COLUMS ALLOWS YOU TO CHECK BOTH YOUR CALCULATIONS
  AND THE AGREEMENT BETWEEN YOUR EXPERIMENTAL DATA AND THE THEORETICAL
  VALUES FOR YOUR SAMPLE.

    ERROR MESSAGE INTERPRETATION:
            *     5  -  10 PERCENT ERROR
           **    10  -  20 PERCENT ERROR
          ***    20  -  40 PERCENT ERROR
            ≠          >40 PERCENT ERROR

          PLEASE HAND IN THIS TABLE WITH YOUR FINAL LAB
   REPORT.  IF YOUR SAMPLE IS AN UNKNOWN, YOU MAY PROCEED
   TO PART 3, WHEN READY, BY TYPING CHEMLAB1 AGAIN.
```

Figure 2. *CHEMLAB*1 Output, Part II

```
[28]  SWITCH←⎕
[29]  →(DATA[VΔ;SWITCH;1]≠14)/DRACULA
[30]  '        SAMPLE DESIGNATION ?-(IE. PURE KCLO3, TYPE PURE)'
[31]  DATA[VΔ;SWITCH;1]←NUMBAR←⎕
[32] DRACULA:→((ι4)∊DATA[I←VΔ;SWITCH;15])/FAR,FURTHER,FINALE,FAREST
[33] FAREST:'        MY RECORDS INDICATE YOU HAVE ALREADY ENTERED VALID INFORMATION FOR
      THE SAMPLE'
[34]  '  TYPE AND TRIAL CHOSEN.  THIS INFORMATION CANNOT BE ALTERED.  IF THERE IS A VAL
      ID REASON TO CHANGE'
[35]  '  THIS DATA, INDICATE IT IN YOUR LABORATORY REPORT.'
[36]  →0
[37] FAR:'  PART 1:  ENTER THE FOLLOWING PIECES OF DATA:'
[38]  ''
[39]  FUDGY←PTR←1
[40]  RETURN←(ɪ26)+1
[41]  '|1|    WEIGHT OF TUBE AND CONTENTS BEFORE HEATING.'
[42]  DATA[I;SWITCH;2]←⎕
[43]  →(ɪ26)+FUDGY
[44]  RETURN←RETURN,(ɪ26)+1
[45]  '|2|    WEIGHT OF TUBE AND RESIDUE AFTER HEATING.'
[46]  DATA[I;SWITCH;3]←⎕
[47]  →(ɪ26)+FUDGY
[48]  RETURN←RETURN,(ɪ26)+1
[49]  '|3|    WEIGHT OF EMPTY TUBE.
[50]  DATA[I;SWITCH;4]←⎕
[51]  →(ɪ26)+FUDGY
[52]  RETURN←RETURN,(ɪ26)+1
[53]  '|4|    TEMPERATURE OF THE OXYGEN [CENTIGRADE]:'
[54]  →((TEMP≥16)∧((TEMP←⎕)≤33))/AROUND
[55]  '       DATA OUTSIDE RANGE OF TEMPERATURE CORRECTION TABLE; 16≤T≤33 °C'
[56]  →RETURN[ρRETURN]
[57] AROUND:DATA[I;SWITCH;5]←TEMP
[58]  →(ɪ26)+FUDGY
[59]  RETURN←RETURN,(ɪ26)+1
[60]  '|5|    BAROMETRIC READING; UNCORRECTED AT ROOM TEMPERATURE.'
[61]  →((TEMP≥710)∧((TEMP←⎕)≤769))/AROUND2
[62]  '       DATA OUTSIDE RANGE OF PRESSURE CORRECTION TABLE; 710≤P≤769 MM. HG'
[63]  →RETURN[ρRETURN]
[64] AROUND2:DATA[I;SWITCH;8]←TEMP
[65]  →(ɪ26)+FUDGY
[66]  RETURN←RETURN,(ɪ26)+1
[67]  '|6|    VOLUME (UNCORRECTED) OF OXYGEN COLLECTED.
[68]  DATA[I;SWITCH;7]←⎕
[69]  →(ɪ26)+FUDGY
[70]  RETURN←RETURN,LAST←(ɪ26)+1
[71]  ''
[72]  '       HAVE YOU MADE ANY ERRORS IN ENTERING YOUR DATA ?'
[73]  →(~⎕)/(PTR=ι3)/GOTCHA,GOAL,GOLLY
[74]  ''
[75]  '       ENTER THE NUMBER OR NUMBERS ( FOUND IN THE | | ) OF THE INPUT STATEMENTS A
      SSOCIATED WITH'
[76]  '  THE DATA YOU WISH TO CORRECT.  IF MORE THAN ONE NUMBER IS TO BE ENTERED, SEPAR
      ATE WITH BLANKS.'
[77]  GOTO←0,(((ι(DQ←ρRETURN)-1)∊⎕),1)/RETURN
[78]  GATE←(2,DQ)ρRETURN,1⌽(RETURN-2)
[79]  →((ρGOTO)≤2)/RETURN[DQ]
[80]  GOTO←1↓GOTO
[81]  FUDGY←(ɪ26)-GATE[2;(GATE[1;]∊GOTO[1])/ιDQ]+2
[82]  →GOTO[1]
[83] GOTCHA:PART←((20×⌊0.5+DATA[I;SWITCH;8]÷20)-700)÷20
[84]  PART←DATA[I;SWITCH;8]-TCTABLE[⌊0.5+DATA[I;SWITCH;5]-15;PART]+
      0.03
[85]  PART←PART-VPTABLE[⌊0.5+DATA[I;SWITCH;5]-15]
[86]  DATA[I;SWITCH;6]←(PERCENT[NUMBAR]×(DATA[I;SWITCH;2]-DATA[I;SWITCH;4])×25528020×(
      DATA[I;SWITCH;5]+273.16))
[87]  DATA[I;SWITCH;6]←DATA[I;SWITCH;6]÷33475.758×PART
[88]  FUDGE←|(DATA[I;SWITCH;6]-DATA[I;SWITCH;7])÷DATA[I;SWITCH;6]
[89]  ''
[90]  →((FUDGE<0.1),((0.1≤FUDGE)∧(FUDGE≤0.2)),((0.2≤FUDGE)∧(FUDGE≤
      0.3)),(FUDGE>0.3))/VG,RG,PG,UA
```

Figure 3. Steps 28-60 of *CHEMLAB1*

Figure 4. Steps 61-90 of *CHEMLAB1*

Line 79: This line is the Question Retry Area termination check. It checks to see if GOTO has fewer than 2 elements; this should occur only after the last student retry request has been processed.

Line 80: This line truncates from GOTO its first element; this element is the statement number associated with the question that has just been processed. On the first pass, this element is set to zero and truncated.

Line 81: This line calculates the value of FUDGY necessary to cause a branch back to Line 79 which is the Question Retry Area termination check. Note that this branch-back will occur from the termination point of the question being retried.

Line 82: This is a branch to the first element of GOTO, which is always a statement number and always a question entry point.

The relatively elaborate data entry and correction features just described make it easy for students new to the computer to use it without frustrating delay. These kinds of features are especially important in the programming of functions to be used by a large number of students within a specified, limited, period of time.

Other features of this program and auxilliary programs necessary for its operation will be presented, as time permits, in the oral presentation.

General Observations and Conclusions

A major program evolves, in general, from many drafts. Problems often seem to arise where no difficulties were anticipated. It is very important that the academic directions and decisions come from the faculty member. It follows from this that a faculty member needs to keep in close touch with the development and debugging of the program, particularly in the early stages. A good, operating, interactive program represents a large investment of programmer and faculty time. Some of this investment may be recovered if an attempt is made to generalize the program so that significant blocks of it can be used in other applications with minor changes.

If the computer is not under your personal control, it is important to consult with the computer center before trying to process a large number of students in a fixed period of time through an interactive program. Languages such as APL are often run simultaneously with other remote languages, and with background batch processing. A low priority assignment to the APL system you are using may result in relatively long delays between the transmission of a line by a student and the response by the computer. Delays of 10 seconds are long, and of 30 seconds or more destructively frustrating. Often a change in priority will reduce such delays to a few seconds at most.

Some students view any kind of computer mediation of insturction as a negative, depersonalizing, undesirable interference with the educational process. It appears to us to be important to emphasize whenever possible that programs such as CHEMLAB are intended to improve student-faculty contact by moving some routine data manipulations to the computer, and using that time saved for dealing with whatever some of the current "real" problems are. In other words, the time spent by the faculty and student in the laboratory is not necessarily reduced, but the questions and discussions which occur seem to be devoted more to "how" and "why" rather than "is the number right?" or "what did I do wrong?".

ACKNOWLEDGMENT

We are particularly indebted to Mr. James Higgins, academic manager of the SUNY-Binghamton computer center, for his encouragement and support of the development of our computer applications in Chemistry. We were very fortunate in having available to us the programming assistance and expertise of Mr. Kevin Kelley and Ms. Anne Kellerman.

FOOTNOTES

1. Thomas R. Dehner and Bruce E. Norcross, "The Use of APL in Computer-Assisted Instruction in Freshman Chemistry"; presented at:

   a. 158th National American Chemical Society Meeting, New York, September 8, 1969;

   b. "Second Conference on Computers in the Undergraduate Curricula", June 23-25, 1971, Dartmouth College, Hanover, New Hampshire;

   c. "Conference on Computers in Chemical Education and Research", July 19-23, 1971, Northern Illinois University, Dekalb, Illinois.

A COLLECTION OF GRAPH ANALYSIS APL FUNCTIONS

E. Girard, D. Bastin and J. C. Rault
Laboratoire Central de Recherches
Thomson-CSF
Domaine de Corbeville, B. P. 10
(91) Orsay, France

Summary

A set of functions dealing with graph theory is presented: graph description, modifications, k-connectivity analysis, and search for paths with given properties. Graph coding coherence and the modularity of APL functions enable one to link these different procedures at will and to use them in such different problems as ~~digital~~ circuit implementation, ~~fault~~ detection, and I. C. mask layout.

The APL functions are given in Appendix 1 and detailed examples in Appendix 2.

1 - Introduction

Graph theory is encountered in many fields of application. Graphs are very useful for modeling and describing processes and systems. Thus there is a constant need for algorithms dealing with graphs and leading to efficient computer programs.

During the development of several projects concerning simulation of digital circuits, generation of fault detection and location sequences, and layout of printed and/or integrated circuits, we had an opportunity to experiment with many graph theoretical algorithms. This led us to write a collection of APL functions which we intend to describe in this paper.

The main data structures and general functions for handling them are first described. Then a collection of APL functions dealing with many aspects of graph theory are detailed.

2 - Graph Description

The choice of a good description for graphs is not a trivial task. Two main constraints, usually conflicting, prevail:

- keeping memory occupation to a minimum,
- providing efficient execution.

Several coding schemes are generally used. On the one hand, there are list structures which meet the first requirement but have the drawback of being unwieldy to handle; on the other hand there is the connection matrix in which the Ith row contains the numbers of the vertices connected (next neighbors, predecessors or successors as the case may be) to vertex numbered I. This coding scheme meets the second requirement but has the objectionable feature of wasting memory unnecessarily.

This state of affairs means that one should not have a single coding scheme but rather the capability of several schemes with the appropriate routines for switching easily from one to the other.

In what follows we use mainly two coding schemes:

* In the first method, akin to list processing, graphs are described by means of a single vector whose components are either zero or integer indices. The indices for the vertices connected to vertex labeled I (adjacent vertices predecessors or successors accordingly) are the components of this vector comprised between the Ith and the I-1th zero. This way of describing graphs is convenient for APL for it keeps memory occupation to a minimum while it is well suited to APL operators. The only disadvantage in certain applications such as graph reduction, is the requirment that vertices be numbered from 1 to N, with N the total number of vertices in the graph under consideration.

* In the second method, graphs are described by means of the so-called arc matrix (or edge matrix in the case of unoriented graphs). This matrix has two columns and as many lines as there are arcs (or edges) in the graph. This scheme leads to memory occupation usually larger than that in the first scheme but it is better fitted to the array capabilities of APL operators.

General Functions

According to the preceding coding schemes it is necessary to handle vectors consisting of groups of indices included between two separators, namely zeros.

V PR N — extracts the Nth group from vector V.

Example:

```
      2 3 0 1 3 0 1 2 4 0 3 0 PR 2
1  3
```

V PP N — gives the components stored in a given vector V between the zero whose index in V is N, and the preceding zero.

Example:

```
      2 3 0 1 3 0 1 2 4 0 3 0 PP 10
1  2  4
```

V SN N — indicates which groups contain a component of value N.

Example:

```
      2 3 0 1 3 0 1 2 4 0 3 0 SN 3
1  2  4
```

V SR N — gives the indices of each 0 immediately following the components of value N in a given vector V

Example:

```
      2 3 0 1 3 0 1 2 4 0 3 0 SR 3
3  6  12
```

V MODIFY W — replaces in the vector V the group of number W[1] by the group formed by W[2], ...XXXXX

Example:

```
      2 3 0 1 3 0 1 2 4 0 3 0 MODIFY 2 5 6 7
2  3  0  5  6  7  0  1  2  4  0  3  0
```

N PERL V — exchanges in vector V components whose values are N and N[2]

Example:

```
      3 4 PERL 2 3 0 1 3 0 1 2 4 0 3 0
2  4  0  1  4  0  1  2  3  0  4  0
```

N PERB V — exchanges in vector V the two blocks whose ranks are N[1] and N[2]

Example:

```
      3 4 PERB 2 3 0 1 3 0 1 2 4 0 3 0
2  3  0  1  3  0  3  0  1  2  4  0
```

Three special functions for sorting and deletion are constantly used. These are:

TTRIC V:	sorts a given vector V in ascending order with deletion of multiple occurences.
GOM V:	deletes multiple occurrences in a given vector V.
TRI V:	merges groups from a describing vector having common components, into a single group.
CODE DECODE V	decodes the vector V according to the code provided by the matrix CODE where the first row corresponds to the new numbering and the second one to the first numbering.

Example:

```
      M
0   1   2   3   4
0   3   2   4   1

      M DECODE 2 3 0 1 3 0 1 2 4 0 3 0
2 1 0 4 1 0 4 2 3 0 1 0
```

4 - Graph-Describing Functions

This niladic function simply builds up, in a conversational mode, the graph describing vector for both oriented and unoriented graphs.

Input: for each vertex the list of its successors (or of the adjacent vertices)

Output: the resulting describing vector.

In the case of unoriented graphs the description is checked for inconsistencies and ill described vertices are printed out.

An example is given in Appendix 2 and will be used throughout this paper for illustrating the different APL functions.

Functions OBTPRE and OBTSUC:

These two complementary monadic functions allow, in the case of oriented graphs, one to obtain the predecessor vector PREDEC from the successor vector SUCCES and vice-versa.

Example:

```
      SUCCES
3 0 3 0 4 0 5 0 6 0 4 7 0 5 8 0 9 0 7 10 0 8 11 0 12
0 13 0 14 15 0 12 0 16 17 0 14 18 0 0 0 20 0 22 0 22 0
11 0

      ⎕←PREDEC←OBTPRE SUCCES
0 0 1 2 0 3 0 0 4 7 0 5 0 6 9 0 7 10 0 8 0 9 0 10 22
  0 11 14 0 12 0 13 16 0 13 0 15 0 [illegible] 0 16 0 0 19 0 0
  20 21 0

        ⎕←OBTSUC PREDEC
  3 0 3 0 [illegible] 0 5 0 6 0 4 7 0 5 8 0 9 0 7 10 0 8 11
  0 12 0 13 0 14 15 0 12 0 16 17 0 [illegible]
  [illegible] 0 11 0
```

Function OBTAJ:

This is for deriving the unoriented graph associated with an oriented graph.

Example:

```
        ⎕←ADJAC←OBTADJ SUCCES
3  0  3  0  1  2  4  0  3  5  0  0  4  6  7  0  4  5  7  0  5  6  8  9
0  7  9  10  0  7  8  10  0  8  9 11  0  10  12  22  0  11  13  14  0
12  14  15  0  12  13  16  0  13  16  17  0  14  15  18  0  15  0
16  0  20  0  19  22  0  22  0  11  20  21  0
```

Functions ARETE and VECT:

The arc matrix ARCMAT or the edge matrix EDGMAT are derived from the corresponding describing vectors SUCCES or ADJAC by means of the function ARETE (French for "edge"):

ARCMAT ← ARETE SUCCES

EDGMAT ← ARETE ADJAC

In fact two different functions are used: one, ARETE, if multiple edges are not considered; the other, AREI, when multiple edges are present.

Conversely, the describing vectors SUCCES and ADJAC are derived from the corresponding matrices:

SUCCES ← VECT ARCMAT

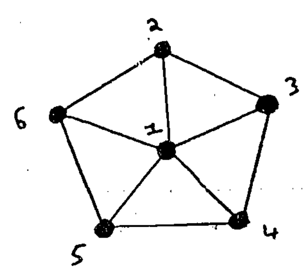
ADJAC ← VECT EDGMAT

Wheels

Function ROUE (French for "wheel") builds up the describing vector ADJAC for a wheel of given order. The "hub" is the vertex labeled 1 and the otehr ones are on the rim.

ADJAC ← ROUE N

Example:




```
        ROUE 5
2  3  4  5  6  0  1  6  3  0  1  4  2  0  1  5  3  0  1  6  4  0
1  5  2  0
```

Adjacent Vertices

Vertices adjacent to a given vertex labeled N are provided by the dyadic function ADJA:

ADJA ← SUCCES ADJA N

Example:

```
      SUCCES ADJA 11
10  12  22

      SUCCES ADJA 1
3
```

Ancestors of a Given Vertex

Function ASCEND gives the set of vertices from which a given vertex labeled N may be reached.

ANC ← SUCCES ASCEND N

Example:

```
      SUCCES ASCEND 22
20  21  19

      SUCCES ASCEND 3
1  2
```

Descendants of a Given Vertex

Conversely the function DESCEN gives the set of vertices which may be reached from vertex labeled N.

DES ← SUCCES DESCEN N

Example:

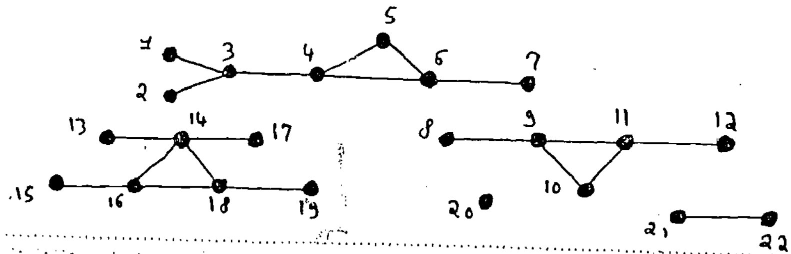
```
      SUCCES DESCEN 12
13  14  15  12  16  17  18

      SUCCES DESCEN 8
9  7  10  5  3  11  6  12  4  13  14  15  16  17  18
```

Connected Components:

Function COMCO derives the weakly-connected component to which the vertex labeled N belongs:

CSC ← ADJAC COMCO N

Example:




```
        ADJAC COMCO 12
12  11  9  10  8


        ADJAC COMCO 20
20
```

Function COFCO in a similar way gives the strongly-connected component to which the vertex labeled N belongs.

CFC ← SUCCES COFCO N

Example:

```
        SUCCES COFCO 3
3
        SUCCES COFCO 7
5  6  8  9  4  7  10
        SUCCES COFCO 10
8  9  7  10  5  6  4
```

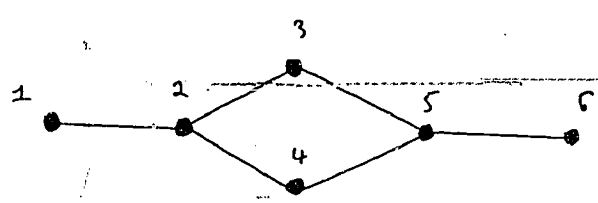
5 - Graph-Structuring Functions

Here are gathered several functions used for modifying graphs by removal, addition or duplication of arcs and/or vertices. The functions given below in this paper concern unoriented graphs only. Similar functions exist for oriented graphs.

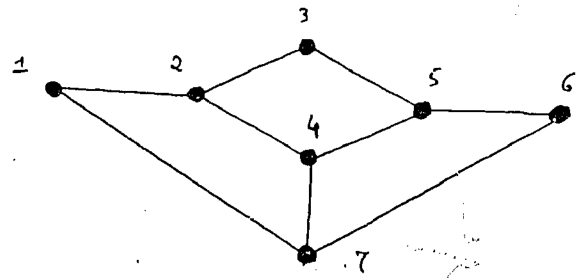
Addition of a Vertex

The added vertex is labeled with an index equal to the highest vertex index in the graph plus one. It is sufficient to indicate by the vector N which of the vertices should be adjacent to the new vertex.

Example:




```
        GRAF2
2  0  1  3  4  0  2  5  0  2  5  0  3  4  0  0  5  0
```

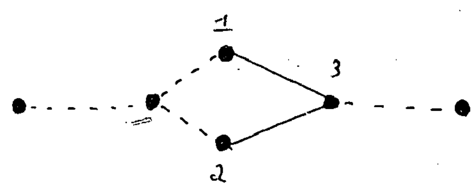


```
      1 4 6 AJOUTS GRAF2
7 2 0 1 3 4 0 2 5 0 7 2 5 0 3 4 6 0 7 5
0 1 4 6 0
```

Deletion of a Vertex

Several vertices may be removed from a graph by means of the function ENLEVS. The vertices to be removed are given in vector N. Vertices are relabeled.

Example:



```
      1 2 6 ENLEVS GRAF2
3 0 3 0 1 2 0
```

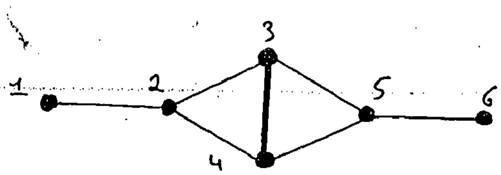
Addition of an Edge

The addition of an edge is performed through the dyadic function AJOUTA:

ADJAC ← N AJOUTA ADJAC

Here N is a two-component vector indicating the two vertices incident to the added edge.

Example:

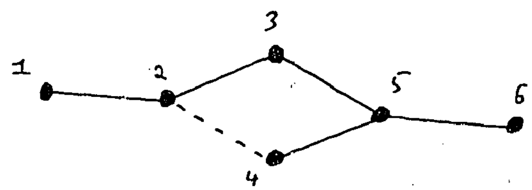


```
      3 4 AJOUTA GRAF2
2 0 1 3 4 0 2 5 4 0 2 5 3 0 3 4 6 0 5 0
```

Deletion of an Edge

Similarly an edge is deleted with the function ENLEVA:

Example:



2 4 *ENLEVA GRAF2*
2 0 1 3 0 2 5 0 5 0 3 4 6 0 5 0

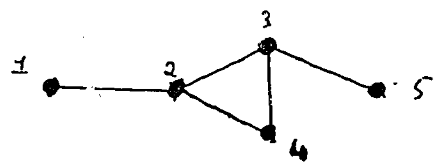
Merging of Two Vertices

On merging two vertices, the resulting vertex is incident to all the edges incident to the two initial vertices. Edges which may link these two vertices are deleted. Vertices are relabeled.

ADJAC ← N CONTRA ADJAC

N is a two-component vector giving indices of vertices to be merged.

Example:



3 5 *CONTRA GRAF2*
2 0 1 3 4 0 2 4 5 0 2 3 0 3 0

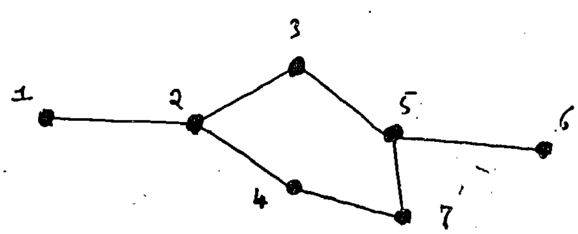
Vertex Splitting

A given vertex may be split so as to generate two vertices. One of these vertices keeps the initial index, the other is labeled with index N + 1 where N is the total number of vertices in the initial graph.

Edges initially incident to the considered vertex are assigned to two resulting vertices according to the user's choice. The way the splitting is performed is fixed in the left argument of the corresponding function dubbed MITOSE for obvious reasons. This left argument is a vector W whose first component has for its value the index for the vertex to be split. The other components are the indices for the vertices which should be kept adjacent to the first resulting vertex.

ADJAC ← W MITOSE ADJAC

Example:



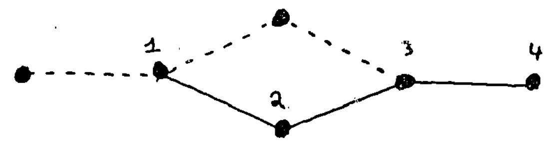
5 3 6 *MITOSE GRAF2*
2 0 1 3 4 0 2 5 0 7 2 0 3 6 7 0 5 0 4 5 0

Maximal Subgraph Extraction

Extracting a maximal subgraph from a graph (which means simply keeping in the graph a set of N vertices along with all the associated edges in the given graph) is performed with the function SSGMAX. Vertices of the subgraph obtained in this way are then relabeled with indices ranging from 1 to N in correspondence with the initial order.

GRAPH ← SG SSGMAX GRAPH

Example:




```
      2 4 5 6 SSGMAX GRAF2
2 0 1 3 0 2 4 0 3 0
```

6 - Graph Analyzing Functions

In this section we give a non-exhaustive set of functions for the determination of the characteristic components of, or the reduction of, graphs.

Determination of In-Degrees and Out-Degrees

The function DEMDEG determines the in- out-degrees of a subset, SET, of vertices in a given graph.

D ← SUCCESS DEMDEG SET

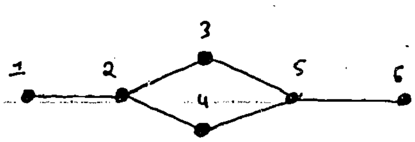
D is a two-component vector where D[1] and D[2] are the in-degree and the out-degree of SET.

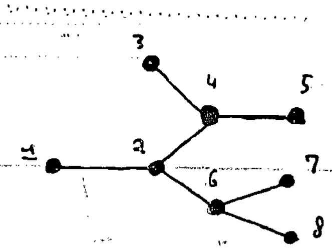
```
      SUCCES DEMDEG 12 13 14 15 16
1 2
```

Checking for Cycles

Cycles in a graph (assumed to be connected) are detected with the function CYCLE. The procedure used here consists in deleting pendent vertices from the graph. Then pendent vertices are deleted from the resulting graph and the procedure is iterated until no more vertices can be deleted. If all vertices have been considered, no cycle is present.

Example:






```
      CYCLE 2 0 1 3 4 0 2 5 0 2 5 0 3 4 6 0 5 0
LE GRAPHE POSSEDE AU MOINS UN CYCLE.
```

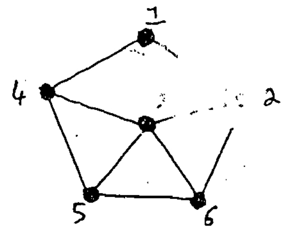
```
      CYCLE 2 0 1 4 6 0 4 0 2 3 5 0 4 0 2 7 8 0 6 0 6 0
LE GRAPHE EST SANS CYCLE.
```

Checking Whether a Graph is a Wheel

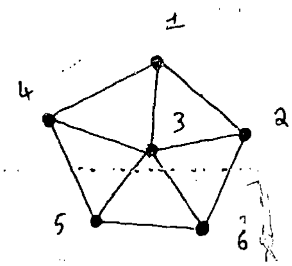
The monadic function WHEEL returns zero if the tested graph is not a wheel, and N if it is a wheel of order N.

N ← WHEEL GRAPH

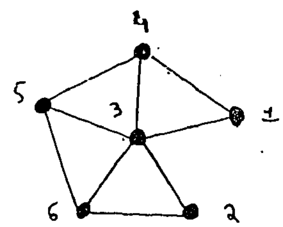
Example:



WHEEL 2 4 0 1 3 6 0 2 4 5 6 0 1 3 5 0 4 3 6 0 5 3 2 0



WHEEL 2 3 4 0 1 3 6 0 1 2 4 5 6 0 1 3 5 0 4 3 6 0 5 3 2 0



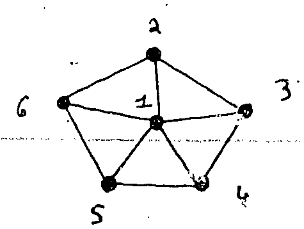
WHEEL 3 4 0 3 6 0 1 2 4 5 6 0 1 3 5 0 4 3 6 0 5 3 2 0

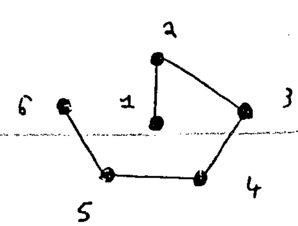
Determination of a Spanning Tree

This is a classical algorithm implemented by function ARBCOV. The result is simply a describing vector for the spanning tree. GRAPH must be renumbered by the function NEWNOT:

SPT ← OLTADJ CODE DECODE ARBCOV NEWNOT GRAPH

Example:





OLTADJ CODE DECODE ARBCOV NEWNOT ROUE 5

2 0 1 3 0 2 4 0 3 5 C 4 6 0 5 0

Determination of Elementary Circuits

Function CIRELM implements an algorithm devised by J. C. Terman[9].

ELCIR ← CIRELM GRAPH

Example:

```
              CIRELM SUCCES
       4  5  6
       5  6  7
       7  8  9
       8  9  10
       12  13  14
       12  13  15  16  14
 4  5  6  0  5  6  7  0  7  8  9  0  8  9  10  0  12  13  14
 0  12  13  15  16  14  0
```

Number of Spanning Trees In A Wheel

Function NSTW returns the number of spanning trees in a wheel of order N.

NUMBER ← NSTW N

The algorithm is described by B. R. Myers[5].

In fact, two different functions are used, one is recurrent (NSTW), the other not (WSTN):

Example:

```
      WSTN 3                    WSTN 3
16                        16
      WSTN 4                    WSTN 4
45                        45
      WSTN 5                    WSTN 5
121                       121
```

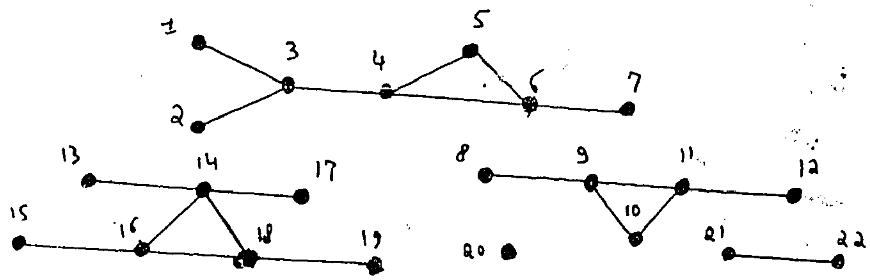
Graph Decomposition Into Connected Components

Reduction of graphs into their weakly-connected components is performed with the function DGCSC which uses a classical procedure. The set of vertices connected to vertex 1 is first derived and extracted from the initial graph, then the procedure is iterated on the remaining graph.

WCCOMP ← DGCSC GRAPH

WCCOMP is a vector formed by the sets of indices of the different components delimited by zeros.

Example:



ADJAC

3 0 3 0 1 2 4 0 3 5 6 0 4 6 0 4 5 7 0 6 0 9 0
8 10 11 0 9 11 0 9 10 12 0 11 0 14 0 13 16 17 18
0 16 0 14 15 18 0 14 0 14 16 19 0 18 0 0 22 0 21 0

DGCCC ADJAC

1 3 2 4 5 6 7 0 8 9 10 11 12 0

13 14 16 17 18 15 19 0 20 0 21 22 0

Graph Decomposition Into Strongly Connected Components

Graphs are reduced in their strongly connected components by the function DGCFC which uses a procedure similar to that described above:

Example:

DGCFC SUCCES

1 0 2 0 3 0 5 6 4 7 8 9 10 0 11 0 13 14 15 12 16
0 17 0 18 0 19 0 20 0 21 0 22 0

Determination of Pendent Vertices

Indices for pendent vertices appear only once in the describing vector

PENVER ← SOMPEN GRAPH

PENVER is a vector whose components are the indices of pendent vertices.

Determination of Rooted Trees

Vertices belonging to rooted trees are determined by considering first pendent vertices and then determining paths from them to the roots. For convenience, roots are not included in their corresponding trees but are considered as articulation points. Their determination is performed by the function REARBO:

input argument:	describing vector
result:	a vector containing the sets of indices of the different rooted trees.

In the following, graphs are assumed to be connected.

Determination of Cycles

Independent cycles are determined while building a spanning tree. Rooted trees are first detected and deleted from the graph:

input data:	vectors describing respectively the graph and its possible rooted trees.
result:	a describing vector for independent cycles whose number is equal to the graph cyclomatic number.

Determination of Lobes

A lobe (2-connected component) is a set of cycles in which two cycles share a common edge. Lobes are determined with the function RELOBE whose input argument is the cycle-describing vector and whose output is the lobe-describing vector.

Determination of Cut-Edges

Cut-edges (bridges) are edges which belong neither to a lobe nor to a rooted tree. Cut-edges sharing a common vertex are considered as a single one.

Determination of Cut-Vertices

This section is concerned with the determination of cut-vertices between two lobes, one lobe and one rooted tree, or one rooted tree and one cut-edge. Cut-vertices within a rooted tree or a cut-edge are not considered here. Function REPOAR is used for this purpose.

For each cut-vertex three sets of data are provided as follows:

a. cut-vertex between two lobes:

1. index of the cut-vertex
2. & 3. ranks of the two lobes in the lobe describing vector.

b. cut-vertex between a lobe and a rooted tree:

1. index for the cut-vertex
2. rank of the lobe in the lobe-describing vector.
3. negative of the rank of the corresponding rooted tree in the rooted-tree-describing vector.

c. Cut-vertex between a rooted tree and a cut-edge:

1. index of the cut-vertex
2. negative of the rank of the corresponding cut-edge in the cut-edge-describing vector
3. rank of the corresponding rooted tree in the rooted-tree-describing vector.

This way of representing cut-vertices (or articulation points) is convenient for finding the components they connect.

Several of the above functions are gathered in a single function, DECOMP, for the reduction and the determination of characteristic elements in a graph.

An example is detailed in Appendix 2.

Eulerian Circuits

Eulerian circuits are determined through a "minirecoil" procedure [7,8] implemented by function EULER.

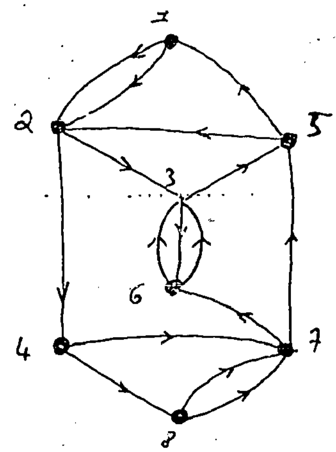
In the case where the graph under consideration is not Eulerian it is modified by duplicating a minimum number of edges in the graph. Function ACHEM performs this operation.

data input for EULER: graph describing vector

result: a message indicating whether the given graph is Eulerian or not.

If not, the list of the paths to be duplicated is printed out. The Eulerian circuit is described by a vector formed by the indices of the vertices, given in the order they are encountered along the circuit.

Example:




```
            EULER L+V
2 2 0 4 3 0 5 6 0 7 8 0 1 2 0 3 3 0 5 6 0 7 7 0
            LE GRAPHE N'EST PAS EULERIEN; CHEMINS AJOUTES :
2 4
3 5 1
7 5 2 4 8
            CIRCUIT EULERIEN :
1 2 4 8 7 5 2 4 8 7 5 2 4 7 6 3 6 3 5 1 2 3 5 1
```

Decomposition Into 2- and 3-Connect Components

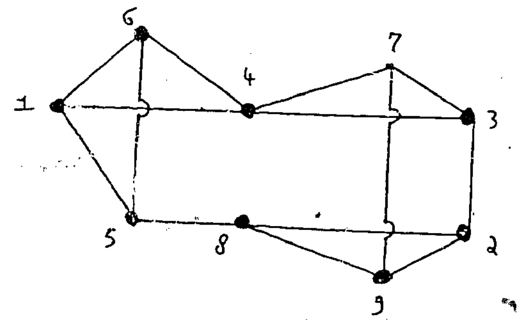
This decomposition takes advantage of results, established by Kleitman[4], for minimizing the number of pairs of vertices for which either two- or three-vertex disjoint paths are sought.

The graph is 2-connected (or 3-connected) if these two (or three) disjoint paths are found. In the case that no such paths exist, the cut-vertex (or cut-set) linking two sets of 2-connected (or 3-connected) components is provided.

This procedure is then iterated on the two resulting sets.

A special labeling procedure is performed to find the vertex disjoint paths[3].

Example:




```
CODE DEC3C ⎕←A1
4 5 6 0 3 8 9 0 2 4 7 0 1 3 5 6 7 0 1 4 6 8 0
  1 4 5 0 3 4 9 0 2 5 9 0 2 7 8 0
COMPOSANTE 3-CONNEXE: 1 4 5 6
COMPOSANTE 3-CONNEXE: 2 3 4 7 8 9
ELEMENT NON 3-CONNEXE: 4 5 8
```

7 - Conclusion

The use of this set of functions, which is constantly undergoing improvement and extension, is illustrated in Appendix 1.

It has been proved to be very useful due to modularity, extensibility and interaction capabilities provided by the APL system.

Interaction is desirable for problems which cannot reasonably be solved in a fully automatic manner. This is the case for problems encountered in graph theory.

Until now, however, in this study APL has been considered more as a tool for establishing rapidly and economically the merits of different algorithms dealing with graphs.

As soon as an algorithm or a set of algorithms are declared suitable, they are turned over to professional programmers for translation into another language (mainly FORTRAN) in order to produce a more efficient program which is of easier access to the whole engineering community.

At the present time this system is intended for development purposes; but with the spread of APL and the imminent availability of computing systems built around APL this situation may be reversed. In this case the use of such design automation tools could be contemplated at any stage in the design process.

The transfer of algorithms from the designer to the engineering program developer usually requires no flowcharting. The APL listing itself is considered here as a reference document.

We feel also that the use of APL language could be extended as a convenient vehicle for communication. We suggest generalizing its use to the formal description of algorithms dealing with graphs.

ACKNOWLEDGMENTS

The authors wihs to express their gratitude to Mr. P. Rosenstiehl of l'Ecole Pratique des Hautes Etudes (PARIS) for his advice and many stimulating discussions, and to the SESCOSEM Company for their financial support.

BIBLIOGRAPHY

1. C. Berge. The Theory of Graphs. Methuen and Co.: London, 1962.

2. C. Berge. Graphes et Hypergraphes (DUNOD), Paris, 1970.

3. I. T. Frisch. "An algorithm for vertex-pair connectivity", Int. Jl. Control, 1967, Vol. 6, No. 6, p. 579-593.

4. D. J. Kleitmann. "Method for investigating connectivity of large graphs", IEEE Trans. on Circuit Theory, May 1969, p. 232-233.

5. B. R. Myers. "Number of spanning trees in a wheel", IEEE Trans. on Circuit Theory, Marcy 1971, p. 280-82.

6. O. Ore. Theory of Graphs. American Mathematical Society: Providence, R. I., 1962.

7. P. Rosenstiehl. Graphes leurs vecteurs et leurs mots. Cours a l'Ecole Polytechnique, Paris, (avec la collaboration de F. Moniez et J. C. Bermond).

8. P. Rosenstiehl. "Labyrinthologie Mathematique" Mathematiques et Sciences Humaines (9eme annee No. 33, 1971, p. 5-32.

9. J. C. Terman. "An efficient search algorithm to find the elementary circuits of a graph", Com. of the ACM, Vol. 13, No. 12, p. 722-726, December 1970.

APPENDIX 1

APL Functions

Statistics showing the frequencies of occurence for the different APL operators is given below. It has been used for a quantitative comparison between APL and FORTRAN programs performing the same operation.

OPERATOR / *OPERATEUR*	NUMBER OF TIMES OPERATOR OCCURS / *NOMBRE D'OCCURRENCES*	FREQUENCY / *FREQUENCE*
←	528	20.5527
,	293	11.4052
/	241	9.38106
[	233	9.06968
+	181	7.04554
ρ	177	6.88984
→	167	6.50058
=	128	4.98248
∨	116	4.51538
ι	68	2.64694
-	63	2.45232
×	58	2.25769
∈	53	2.06306
~	37	1.44025
:	35	1.3624
<	33	1.28455
>	31	1.2067
≠	31	1.2067
⌈	15	0.583885
≥	14	0.544959
∧	12	0.467108
⎕	12	0.467108
-	11	0.428182
.	7	0.27248
∨	5	0.194628
⌊	5	0.194628
⌽	3	0.116777
⍉	3	0.116777
\	2	0.0778513
|	2	0.0778513
÷	1	0.0389257
≤	1	0.0389257
⍞	1	0.0389257
∘	1	0.0389257
⌿	1	0.0389257
⋆	0	0
⍟	0	0
!	0	0
⍀	0	0
⊙	0	0
⊥	0	0
⊤	0	0

```
     ∇ACHEM;D;DN;V;H;I;J
[1]   D←⍳I←0
[2]   →(2+I=⌈/M[;1]),D←D,(+/M[;2]=I)-+/M[;1]=I←I+1
[3]   →(∧/D=0)/14
[4]   (1 5)ρ' ';'LE GRAPHE N''EST PAS EULERIEN; CHEMINS AJOUTES :'
[5]   DN←(D<0)/⍳ρD
[6]   V←(H←(D>0)/⍳ρD),J←0
[7]   →(∨/DN∊H←TTRIC(M[;1]∊H)/M[;2])/9
[8]   →7,V←H,0,V
[9]   H←I←((DN∊H)/DN)[1]
[10]  →(10+J=+/V=0),H←(I←((I∊V PR J←J+1)/I←(I=M[;2])/M[;1])[1]),H
[11]  D[H[1,ρH]]←(D[H[1]]-1),D[H[ρH]]+I←1
[12]  →(12+(I←I+1)=ρH),,M←((1+(ρM)[1]),2)ρ(,M[⍳J;]),H[I],H[I+1],,M[J+⍳(ρM)[1]-J←¯1+M[;1]⍳H[I];]
[13]  →(5×~∧/0=D),T←T,(¯1+ρ⎕←H)ρ0
[14]  (1 5)ρ' ';'LE GRAPHE EST EULERIEN.'
[15]  ∇

     ∇Z←V ADJA N
[1]   Z←TTRIC(V PR N),V SN N
[2]   ∇

     ∇Z←N AJOUTA V;A;B
[1]   →(2×~0∊N),Z←V
[2]   Z←((V[B]-1)ρZ),A,((¯1+V[B])DROP(V[A]-1)ρZ),(B←⌊/N),(¯1+(V←(Z=0)/⍳ρZ)[A←⌈/N])DROP Z
[3]   ∇

     ∇Z←N AJOUTS V;I;K
[1]   K←1++/V=I←0
[2]   →((I←I+1)>ρN)/4
[3]   →2,V←V MODIFY N[I],K,V PR N[I]
[4]   Z←V,N,0
[5]   ∇

     ∇Z←ARBCOV W;I;A
[1]   I←1+Z←0
[2]   Z←Z,(⌈/(A<I)/A←W PR I←I+1),0
[3]   →2×I<+/W=0
[4]   ∇

     ∇M←ARET V;A;I;F
[1]   F←+/V=I←ρM←⍳0
[2]   →(2+I≥F),M←M,(I×A=0)+A←((2×ρA)ρ0,1)\A←V PR I←I+1
[3]   M←((-F-ρV),2)ρM
[4]   ∇

     ∇M←ARETE V;A;B;I;F
[1]   F←+/V=I←ρM←⍳0
[2]   →(2+I≥F),M←M,(I×B=0)+B←((2×ρA)ρ0,1)\A←TTRIC V PR I←I+1
[3]   M←((-F-ρV),2)ρM
[4]   ∇

     ∇Z←V ASCEND N;I;A
[1]   →(0=ρZ←V SN N)/I←0
[2]   →2×I≠ρZ←Z,(~A∊Z)/A←V SN Z[I←I+1]
[3]   ∇
```

```
      ∇Z←NC CHEMIN V;J;T;TT;SME;SMS;VOIS;I;L
[1]   →(NC≥+/V=0)/ρZ←L←⍳0
[2]   I←I⍳⌊/I←+/M[;1]∘.=⍳⌈/(M←ARETE V)[;1+J←0]
[3]   MARK←((ρSFLOT),2)ρAFLOT←(ρM)[1]ρSFLOT←(⌈/M[;1])ρT←TT←0
[4]   →((J>ρSFLOT),I=J←J+1)/9 4
[5]   MARQUE I,J
[6]   →(T=TT)/11
[7]   TT←T
[8]   →((NC=TT),1)/3 5
[9]   →10×0≠NC←NC-1
[10]  →2,(L←L,I++/L≤I),V←I ENLEVS V
[11]  SME←(MARK[;1]≠0)/⍳(ρMARK)[1]
[12]  SMS←(MARK[;2]≠0)/⍳(ρMARK)[1]
[13]  Z←TTRIC((~SME∊SMS)/SME),(~VOIS∊SME)/VOIS←(M[;1]∊SMS)/M[;2]
[14]  →(0=ρL)/0
[15]  Z←2ρL,Z←((1≠ρL)×Z≥⌈/L)+Z←Z+Z≥⌊/L
[16]  ∇

      ∇Z←CIRELN G;P;H;I;J;K;GPRK
[1]   P←Nρ(K←1),H←(N←+/G=0)ρJ←ρZ←⍳0
[2]   →((GPRK[J]>P[1])∧(~GPRK[J]∊P)∧~(GPRK←(G PR P[K]),0)[J←J+1]∊H PR P[K])/10
[3]   →(J<ρGPRK)/2
[4]   →(~P[1]∊GPRK)/6
[5]   Z←Z,(L←(P≠0)/P),0
[6]   →(K=1)/12
[7]   H←H MODIFY,P[K]
[8]   H←H MODIFY(P[K-1]),(H PR P[K-1]),P[K]
[9]   →2,K←K-1+J←P[K]←0
[10]  K←K+1
[11]  →2,(J←0),P[K]←(G PR P[K-1])[J]
[12]  →(P[1]=N)/0
[13]  →2,(P[1]←P[1]+K←1),H←NρJ←0
[14]  ∇

      ∇Z←V COFCO N
[1]   Z←((~N∊Z)/N),Z←(Z∊V ASCEND N)/Z←V DESCEN N
[2]   ∇

      ∇Z←V COMCO I;K;A
[1]   Z←,I←K←0
[2]   →(K<ρZ←Z,(~A∊Z)/A←V PR Z[K←K+1])/2
[3]   ∇

      ∇Z←N CONTRA V;A
[1]   V←V MODIFY N[1],((A≠N[2])/A←V PR N[1]),(A≠N[1])/A←Z←V PR N[2]
[2]   Z←((A-1+ρZ)ρV),(A←((V=0)/⍳ρV)⌊N[2])DROP V←(V×V≠N[2])+(N[1]×V=N[2])-V>N[2]
[3]   ∇

      ∇CYCLE V;T;W
[1]   W[W⍳T←⍳⌈/W←V]←0
[2]   →3+0=ρT←(~T∊W)/T
[3]   →1+3×1≥ρV←T ENLEVS V
[4]   (25×0=ρT)DROP(25+36×0=ρT)ρ'LE GRAPHE EST SANS CYCLE.LE GRAPHE POSSEDE AU MOINS UN CYCLE.'
[5]   ∇

      ∇Z←M DECODE V
[1]   Z←M[1;M[2;]⍳V]
[2]   ∇
```

```
     ∇DECOMP V
[1]  ''
[2]  '                    ***************************************'
[3]  '                    *  ELEMENTS REMARQUABLES DU GRAPHE   *'
[4]  '                    ***************************************'
[5]  ''
[6]  ARBO←REARBO V
[7]  LOBES←RELOBE RECYCL V
[8]  PONTS←REPONT V
[9]  POAR←REPOAR V
[10] ''
[11] '           **********************************************************'
[12] '           *  ENCHAINEMENT DES ELEMENTS REMARQUABLES DU GRAPHE   *'
[13] '           **********************************************************'
[14] ''
[15] DECRIR
[16] ∇

     ∇DECRIR;I;A;V;W1;W2
[1]  I←0
[2]  E1:→((I←I+1)>+/POAR=0)/E2
[3]  →((A[2]<0),(A←POAR PR I)[3]<0)/E9,E3
[4]  W1←' AU LOBE '
[5]  →E4,W2←LOBES PR A[3]
[6]  E3:W1←' A L''ARBORESCENCE ';W2←ARBO PR-A[3]
[7]  E4:→E1,⎕←'LE POINT D''ARTICULATION ';A[1];' RELIE LE LOBE ';LOBES PR A[2];W1;W2
[8]  E9:→E1,⎕←'LE POINT D''ARTICULATION ';A[1];' RELIE LE PONT ';PONTS PR-A[2];' A L
                                                       ''ARBORESCENCE ';ARBO PR A[3]
[9]  E2:I←0
[10] E5:→((I←I+1)>+/PONTS=0)/ρW1←⍳V←0
[11] A←PONTS PR I
[12] E6:→((V←V+1)>ρA)/E7
[13] →E6,W1←W1,(~W2∊W1)/W2←LOBES UN A[V]
[14] E7:V←1
[15] E8:→((V←V+1)>ρW1)/E5
[16] →E8,⎕←'LE PONT ';A;' RELIE LE LOBE ';LOBES PR W1[1];' AU LOBE ';LOBES PR W1[V]
[17] ∇

     ∇CK DEC2C V;I;A;K;C1
[1]  →((I←0)=ρ,A←2 CHEMIN V)/5
[2]  C1←C1+(C1←DGCSC A ENLEVS V)≥A
[3]  CK[K]DEC2C(K←TTRIC A,C1 PR I←I+1)SSGMAX V
[4]  →3×I<+/C1=0
[5]  (22×2≥ρCK)DROP(22+23×2≥ρCK)ρ'COMPOSANTE 2-CONNEXE: ELEMENT NON 2-CONNEXE: ';CK
[6]  ∇

     ∇CK DEC3C V;I;A;K;C1
[1]  →((I←0)=ρ,A←3 CHEMIN V)/5
[2]  C1←(C1≥⌈/A)+C1←C1+(C1←DGCSC A ENLEVS V)≥⌊/A
[3]  CK[K]DEC3C(K←TTRIC A,C1 PR I←I+1)SSGMAX(A×~A[1]∊V PR A[2])AJOUTA V
[4]  →3×I<+/C1=0
[5]  (22×3≥ρCK)DROP(22+23×3≥ρCK)ρ'COMPOSANTE 3-CONNEXE: ELEMENT NON 3-CONNEXE: ';CK
[6]  ∇

     ∇Z←DEGRAF;ORIENT;I;SOM;FIN;J;SOMMET;A
[1]  20ρ' ';23ρ'*'
[2]  20ρ' ';'*DESCRIPTION DU GRAPHE*'
[3]  20ρ' ';23ρ'*'
[4]  ''
[5]  ⍞←'LE GRAPHE EST-IL ORIENTE ?  (REPONDRE OUI OU NON)'
[6]  DON1:ORIENT←3ρ⍞
[7]  →((∧/ORIENT='OUI')∨∧/ORIENT='NON')/SUITE1
[8]  COR1:→DON1,⎕←'MAUVAISE REPONSE, RECOMMENCER'
[9]  SUITE1:ORIENT←∧/ORIENT='OUI'
[10] ''
[11] 20ρ' ';23ρ'*';(4×1-ORIENT)ρ'*';8ρ'*'
[12] 20ρ' ';'*DESCRIPTION DU GRAPHE ';(4×1-ORIENT)ρ'NON ';'ORIENTE*'
[13] 20ρ' ';23ρ'*';(4×1-ORIENT)ρ'*';8ρ'*'
[14] ''
[15] 'LISTE DES ';(12×ORIENT)ρ'SUCCESSEURS ';(18×1-ORIENT)ρ'SOMMETS ADJACENTS '
```

```
[16]  ''
[17]  '                          INSTRUCTIONS'
[18]  '                         --------------'
[19]  'EN CAS D''ERREUR DE DESCRIPTION POUR LE SOMMET I,TAPER:  SOMMET;I  A LA PLACE'
[20]  'DE LA DESCRIPTION D''UN SOMMET ULTERIEUR.'
[21]  'POUR TERMINER LA DESCRIPTION TAPER: FIN'
[22]  '                        ---------------'
[23]  ''
[24]  Z←⍳FIN←SOMMET←I←J←0
[25]  SUITE2:'SOMMET ';I←I+1
[26]  SOM←,⎕
[27]  →(0≠⍴SOM)/TEST1
[28]  →MODIF×⍳J
[29]  →SUITE2,Z←Z,0
[30]  TEST1:→((1=⍴SOM)∧SOM[1]=0)/TESTF
[31]  →(SOM[1]=0)/SUITE3
[32]  →MODIF×⍳J
[33]  →SUITE2,Z←Z,SOM,0
[34]  SUITE3:J←1
[35]  I←SOM[2]-1
[36]  →SUITE2
[37]  MODIF:Z←Z MODIFY,I,SOM
[38]  J←0
[39]  I←+/Z=0
[40]  →SUITE2
[41]  ''
[42]  TESTF:'NOMBRE DE SOMMETS DU GRAPHE : ';I-1
[43]  →(ORIENT=1)/0
[44]  TESTCO:A←VERIFY Z
[45]  →(0=⍴A)/I←0
[46]  SUITE5:'SOMMET ';A[I←I+1]
[47]  Z←Z MODIFY A[I],⎕
[48]  →(I<⍴A)/SUITE5
[49]  →TESTCO
[50]  ∇

      ∇Z←V DEMDEG A;I
[1]   Z←2⍴I←0
[2]   →(2×I<⍴A),(Z[2]←Z[2]++/~(V PR A[I])∊A),Z[1]←Z[1]++/~(V SN A[I←I+1])∊A
[3]   ∇

      ∇Z←V DESCEN N;I;A
[1]   →(0=⍴Z←V PR N)/I←0
[2]   →2×I≠⍴Z←Z,(~A∊Z)/A←V PR Z[I←I+1]
[3]   ∇

      ∇Z←DGCFC V;I
[1]   Z←⍳I←0
[2]   →((I>+/V=0),(I←I+1)∊Z)/0 2
[3]   →2,Z←Z,(V COFCO I),0
[4]   ∇

      ∇Z←DGCSC V;I
[1]   Z←⍳I←0
[2]   →((I>+/V=0),(I←I+1)∊Z)/0 2
[3]   →2,Z←Z,(V COMCO I),0
[4]   ∇

      ∇Z←N DROP V
[1]   Z←((N⍴0),(((⍴V)-N)⍴1))/V
[2]   ∇

      ∇Z←N ENLEVA V;A
[1]   Z←(V MODIFY N[1],(A≠A[A⍳N[2]]←0)/A←V PR N[1])MODIFY N[2],(A≠A[A⍳N[1]]←0)/A←V PR N[2]
[2]   ∇

      ∇Z←N ENLEVS V;I
[1]   Z←(~V∊N←TTRIC,N)/V+I←0
[2]   →(2×I≠⍴N←N-1),Z←((V-1+⍴Z PR N[I])⍴Z),(V←((Z=0)/⍳⍴Z)[N[I]])DROP Z←Z-N[I←I+1]<Z
[3]   ∇
```

```
      ∇Z←EULER V;I;T;M;L
[1]   T←(ρM←ARET V)[L←Z←,1]ρ0
[2]   ACHEM
[3]   →(~0∊T[I←(Z[1]=M[;2])/⍳(ρM)[1]])/5
[4]   →3,(Z[2]←-Z[2]),(T[I]←1),L←L,(Z←M[I←((0=T[I])/I)[1];1],Z)[1]
[5]   →(1=ρL)/8
[6]   Z←L[I←¯1+ρL],Z
[7]   →3,L←IρL
[8]   (1 5)ρ' ';'CIRCUIT EULERIEN :
[9]   Z←⌽(Z>0)/Z
[10]  ∇

      ∇Z←GOM V;K
[1]   →(0=ρV←,V)/ρZ←⍳K←0
[2]   →(2×K<ρV),Z←Z,(~V[K]∊Z)/V[K←K+1]
[3]   ∇

      ∇Z←M INDIC V
[1]   Z←(M∧.=V)/⍳(ρM)[1]
[2]   ∇

      ∇Z←LS N
[1]   →(N=1 2)/3 4
[2]   →0,Z←(LS N-1)+LS N-2
[3]   →0,Z←1
[4]   Z←3
[5]   ∇

      ∇MARQUE V;SM;MT;R;RT;P;I
[1]   MARK[SM;2]←SM←V[,I←1]
[2]   MT←⍉(7,ρR)ρ(,⍉M[R;]),(,⍉MARK[M[R;2];]),SFLOT[M[R;2]],AFLOT[(R←(M[;1]=SM[I])/⍳(ρM)[1]),
[3]   →((P←MARK[SM[I];]≠0)[2]=0)/5                                          RT←(M[;2]=SM[I])/⍳(ρM)[1]]
[4]   →(5+P[1]=0),MT[;3]←MT[;3]+MT[;1]×∧/MT[;3 6]=0
[5]   MT[;4]←MT[;4]+MT[;1]×</MT[;4 7]≠0
[6]   →7,(SM←SM,MT[((∨/MARK[M[R;2];]≠MT[;3 4])/⍳(ρMT)[1]);2]),(MT[;3]←MT[;3]+MT[;2]×2=-/MT[;
MT[;4 3 5]=0
[7]   →(8+∨/MARK[V[2];]≠0),,MARK[M[R;2];]←MT[;3 4]          4 3 5]≠0),MT[;4]←MT[;4]+MT[;2]×2=-/
[8]   →(2+7×(ρSM)<I←I+1)
[9]   P←I←V[2]
[10]  →(0=ρI←(~MARK[I;]∊0,I)/MARK[I;])/0
[11]  →(10+2×I=V[1]),P←P,I←I[1]
[12]  T←T+I←1,0ρSFLOT[I]←~SFLOT[I←(~P∊V)/P←⌽P]
[13]  →(AFLOT[R←M INDIC V←P[I,I+1]]=1)/16
[14]  →(AFLOT[RT←M INDIC⌽V]=0)/16
[15]  →17,AFLOT[R,RT]←SFLOT[V]←0
[16]  AFLOT[R]←~AFLOT[R]
[17]  →((ρP)>I←I+1)/13
[18]  MARK←(ρMARK)ρ0
[19]  ∇

      ∇Z←W MITOSE V;W1;U;N;I
[1]   Z←(V MODIFY W1,W,N←1++/V=I←0),(U←(~U∊W←W[1+⍳¯1+ρW])/U←V PR W1),(W1←W[1]),0
[2]   →((I←I+1)>ρU)/0
[3]   →2,Z←Z MODIFY U[I],N,(W≠W1)/W←V PR U[I]
[4]   ∇

      ∇Z←W MODIFY NV;M
[1]   Z←((M-1+ρW PR NV[1])ρW),(1 DROP NV),0,(M←((W=0)/⍳ρW)[NV[1]])DROP W
[2]   ∇

      ∇Z←NEWNOT W;I;J
[1]   Z←CODE←,W;I←1
[2]   Z←(I,J)PERL(I,J←((Z≥I←I+1)/Z)[1])PERB Z
[3]   CODE←(I,J)PERB CODE
[4]   →(I<+/Z=0)/2
[5]   CODE←(2,1+ρCODE)ρ0,(CODE←GOM(CODE≠0)/CODE),0,GOM(Z≠0)/Z
[6]   ∇

      ∇Z←NSTW N
[1]   Z←((LS N)*2)+4×¯1+2|N
[2]   ∇
```

```
      ∇Z←OBTADJ  V;I
[1]   Z←⍳I←0
[2]   →(2×I<+/V=0),Z←Z,(TTRIC(V PR I),V SN I←I+1),0
[3]   ∇

      ∇Z←OBTPRE  V;I
[1]   Z←⍳I←0
[2]   →(2×I<+/V=0),Z←Z,(V SN I←I+1),0
[3]   ∇

      ∇Z←OBTSUC  V;I
[1]   Z←⍳I←0
[2]   →(2×I<+/V=0),Z←Z,(V SN I←I+1),0
[3]   ∇

      ∇Z←N PERB W;V
[1]   V←(W=0)/⍳ρZ←W
[2]   →(1=ρN←TTRIC N)/0
[3]   Z←(V[N[1]-1]ρW),(W PR N[2]),0,((V[N[2]-1]-V[N[1]])ρV[N[1]] DROP W),(W PR N[1])
                                                                   ,0,V[N[2]] DROP W
[4]   ∇

      ∇Z← N PERL W
[1]   Z←W
[2]   →(N[1]=N[2])/0
[3]   Z←(N[2]×W=N[1])+(N[1]×W=N[2])+W×~W∊N
[4]   ∇

      ∇W←V PP N;Z
[1]   →(((Z←(V=0)/⍳ρV)⍳N)≠1)/3
[2]   →0,W←(N-1)ρV
[3]   W←Z[(Z⍳N)-1]DROP(N-1)ρV
[4]   ∇

      ∇W←V PR N;Z
[1]   →(N≠1)/3
[2]   →0,W←(((V=0)/⍳ρV)[1]-1)ρV
[3]   W←Z[N-1]DROP((Z←(V=0)/⍳ρV)[N]-1)ρV
[4]   ∇

      ∇Z←REARBO V;W;I;A;B;K
[1]   W←SOMPEN V,Z←⍳I←K←0
[2]   →((K←K+1)>ρW)/6
[3]   →(0=ρB←(~B∊W)/B←V PR W[K])/2
[4]   →(1<ρ(~A∊W)/A←V PR B)/2
[5]   →2,W←W,B
[6]   →((I←I+1)>ρW)/K←0
[7]   →((B←,W[I])∊Z)/6
[8]   →(8+K=ρB),B←B,(~A∊B)/A←(A∊W)/A←V PR B[K←K+1]
[9]   →6,(Z←Z,B,0),⍞←'ARBORESCENCE : ';B
[10]  ∇

      ∇Z←RECYCL V;K;I;A;N;NC;F;ARB;AA;TREE
[1]   →(0=NC←1+((+/V≠0)÷2)-+/V=0)/ρZ←TREE←⍳N←I←K←0
[2]   A←(~A∊ARBO)/A←V PR ARB←,((~A∊F←(ARBO≠0)/ARBO)/A←⍳+/V=0)[1]
[3]   →(A[I←I+1]∊ARB)/5
[4]   →3,(I←0),(A←(~A∊F,ARB[¯1+ρARB])/A←V PR A[I]),ARB←ARB,A[I]
[5]   →((NC>N←N+1)×6-3×I<ρA),(Z←Z,AA,0),⍞←'CYCLE : ';AA←(¯1+ARB⍳A[I])DROP ARB
[6]   →(0=ρA←(~A∊ARB,F,TREE)/A←V PR ARB[(ρARB)-K←K+1])/0
[7]   →3,(I←K←0),(ARB←((ρARB)-K)ρARB),TREE←TREE,KρΦARB
[8]   ∇
```

```
      ∇Z←RELOBE V;K;J;U;W
[1]   K←0
[2]   K←K+1
[3]   →((J←K)>+/V=0)/10
[4]   W←V PR K
[5]   →((J←J+1)>+/V=0)/2
[6]   →(2>+/W∊L←V PR J)/5
[7]   →8+K=1
[8]   →3,V←(Z[K-1]⍴V),(Z[K]DROP Z[J-1]⍴V),(CON W,U),((Z←(V=0)/⍳⍴V)[J]-1)DROP V
[9]   →3,V←(Z[1]DROP Z[J-1]⍴V),(CON W,U),((Z←(V=0)/⍳⍴V)[J]-1)DROP V
[10]  Z←V+K←0
[11]  →((K←K+1)>+/Z=0)/0
[12]  →11,⎕←'LOBE : ';Z PR K
[13]  ∇

      ∇Z←REPOAR V;A;I;K;B;C;J
[1]   Z←⍳I←K←0
[2]   →((I←I+1)=+/LOBES=0)/7
[3]   A←LOBES PR K←I
[4]   →((K←K+1)>+/LOBES=0)/2
[5]   →(1≠⍴B←(A∊C←LOBES PR K)/A)/4
[6]   →4,Z←Z,B,I,K,0
[7]   I←0
[8]   →((I←I+1)>+/ARBO=0)/K←0
[9]   A←ARBO PR I
[10]  →((K←K+1)>⍴A)/8
[11]  →(0=⍴B←(~B∊ARBO)/B←V PR A[K])/10
[12]  →((J←0)=⍴C←LOBES SN B)/15
[13]  →((J←J+1)>⍴C)/8
[14]  →13,Z←Z,B,C,-J,0
[15]  →8,Z←Z,B,(-PONTS SN B),I,0
[16]  ∇

      ∇Z←REPONT V;I;A;B;K;U
[1]   Z←⍳I←0
[2]   →((I←I+1)>+/V=0)/9
[3]   →(I∊ARBO)/2
[4]   →(0=⍴B←LOBES SN I)/2
[5]   U←⍳K←0
[6]   →(6+K=⍴B),U←U,LOBES PR B[K←K+1]
[7]   →(0=⍴A←(~A∊ARBO,U)/A←V PR I)/2
[8]   →2,Z←Z,I,A,0
[9]   Z←TRI Z
[10]  ∇

      ∇Z←ROUE N;I
[1]   Z←(1+⍳N),0,1,(N+1),(1+I←2),0
[2]   →(2+I=N),Z←Z,1,(1+I←I+1),I,0
[3]   Z←Z,1,I,2,0
[4]   ∇

      ∇H←SL N;Z;I
[1]   Z←1,3;I←2
[2]   Z←Z,Z[I-2]+Z[¯1+I←I+1]
[3]   →2×⍳I<N
[4]   H←Z[N]
[5]   ∇

      ∇Z←V SN N;I;A
[1]   →(0=⍴A←(V=N)/⍳⍴V)/⍴Z←⍳I←0
[2]   →(2×I<⍴A),Z←Z,1++/0=V[⍳A[I←I+1]]
[3]   ∇

      ∇Z←SOMPEN V
[1]   V[V⍳⌈/V]←0
[2]   Z←(~(⍳⌈/V)∊V)/⍳⌈/V
[3]   ∇
```

100

```
      ∇Z←V SR N;U;A
[1]   →(0=ρA←(V=N)/⍳ρV)/ρZ←⍳0
[2]   →(2×0≠ρA←(~A∊U)/A),Z←Z,U←(0=V[A])/A←A+1
[3]   ∇

      ∇Z←G SSGMAX V;I
[1]   V←(V∊G,0)/V,Z←⍳I←0
[2]   →((I←I+1)∊G)/4
[3]   →(2×I<⌈/G),(V←V-V>+/Z=0),Z←Z-Z>+/Z=0
[4]   →(2×I<⌈/G),Z←Z,(V PR I),0
[5]   ∇

      ∇Y←TRI V;K;J;Z;W;U
[1]   Y←V+K←0
[2]   K←K+1
[3]   →((J←K)>+/Y=0)/0
[4]   W←Y PR K
[5]   →((J←J+1)>+/Y=0)/2
[6]   →(0=∨/W∊U←Y PR J)/5
[7]   →(K=1)/9
[8]   →3,Y←(Z[K-1]ρY),(Z[K]DROP Z[J-1]ρY),(GOM W,U),((Z←(Y=0)/⍳ρY)[J]-1)DROP Y
[9]   →3,Y←(Z[1]DROP Z[J-1]ρY),(GOM W,U),((Z←(Y=0)/⍳ρY)[J]-1)DROP Y
[10]  ∇

      ∇Z←TRIC W;K
[1]   →(0=ρZ←W←,W)/K←0
[2]   →(2×K<ρW),Z←((Z<W[K])/Z),((Z=W[K])/Z),(Z>W[K←K+1])/Z
[3]   4    ⊃IH⍟
CHARACTER ERROR
      4    ⊃IH⍟
           ^
[3]    )CLEAR
       )CARD
      ∇Z←TRIC W;K
[1]   →(0=ρZ←W←,W)/K←0
[2]   →(2×K<ρW),Z←((Z<W[K])/Z),((Z=W[K])/Z),(Z>W[K←K+1])/Z
[3]   ∇

      ∇Z←TTRIC W;K
[1]   →(0=ρZ←W←,W)/K←0
[2]   →(2×K<ρW),Z←((Z<W[K])/Z),W[K],(Z>W[K←K+1])/Z
[3]   ∇

      ∇Z←VECT M;I;A
[1]   A←⌈/(,M),Z←⍳I←0
[2]   →(2×I<A),Z←Z,((M[;1]=I←I+1)/M[;2]),0
[3]   ∇

      ∇Z←VERIFY W;I;V1;V2
[1]   Z←⍳I←0
[2]   A0:→(∧/(V1←W PR I)∊V2←W SN I←I+1)/A1
[3]   Z←Z,I,(~V1∊V2)/V1
[4]   A1:→(∧/V2∊V1)/A2
[5]   Z←Z,I,(~V2∊V1)/V2
[6]   A2:→(I<+/W=0)/A0
[7]   →(0=ρZ←TTRIC Z)/A3
[8]   →0,⎕←'LA DESCRIPTION DES SOMMETS ADJACENTS AUX SOMMETS  ';Z;' EST INEXACTE'
[9]   A3:'LA DESCRIPTION DU GRAPHE EST COHERENTE.'
[10]  ∇

      ∇Z←WHEEL V;I;N
[1]   N←+/V=I←Z←0
[2]   →((N<I),(3=+/V=I←I+1),Z≠0)/5 2 4
[3]   →2×0≠Z←(N-1)×(N-1)=+/V=I
[4]   →Z←0
[5]   Z←(3×(N=4)×Z=0)+(Z≠0)×N-1
[6]   ∇

      ∇Z←WSTN N
[1]   Z←((SL N)*2)+4×¯1+2|N
[2]   ∇
```

APPENDIX 2

EXAMPLES

```
SUCCES+DEGRAF
                    ***********************
                    *DESCRIPTION DU GRAPHE*
                    ***********************

LE GRAPHE EST-IL ORIENTE ?  (REPONDRE OUI OU NON)
OUI

                    *******************************
                    *DESCRIPTION DU GRAPHE ORIENTE*
                    *******************************

LISTE DES SUCCESSEURS

                    INSTRUCTIONS
                    ------------
EN CAS D'ERREUR DE DESCRIPTION POUR LE SOMMET I,TAPER:  SOMMET,I  A LA PLACE
DE LA DESCRIPTION D'UN SOMMET ULTERIEUR.
POUR TERMINER LA DESCRIPTION TAPER: FIN
                    ------------

SOMMET 1
⎕:
      3
SOMMET 2
⎕:
      3
SOMMET 3
⎕:
      4
SOMMET 4
⎕:
      5
SOMMET 5
⎕:
      6
SOMMET 6
⎕:
      4 7
SOMMET 7
⎕:
      5 8
SOMMET 8
⎕:
      9
SOMMET 9
⎕:
      7 10
SOMMET 10
⎕:
      8 11
SOMMET 11
⎕:
      12
SOMMET 12
⎕:
      13
```

```
SOMMET 13
⎕:
      14 15
SOMMET 14
⎕:
      12
SOMMET 15
⎕:
      16 17
SOMMET 16
⎕:
      14 18
SOMMET 17
⎕:
      10
SOMMET 18
⎕:
      10
SOMMET 19
⎕:
      20
SOMMET 20
⎕:
      22
SOMMET 21
⎕:
      22
SOMMET 22
⎕:
      11
SOMMET 23
⎕:
      FIN
NOMBRE DE SOMMETS DU GRAPHE : 22


      SUCCES
3 0 3 0 4 0 5 0 6 0 4 7 0 5 8 0 9 0 7 10 0 8 11 0 12
    17 0 14 18 0 0 0 20 0 22 0 22 0 11 0

                                        0 13 0 14 15 0 12 0 16


      ⎕←PREDEC←OBTPRE SUCCES
0 0 1 2 0 3 6 0 4 7 0 5 0 6 9 0 7 10 0 8 0 9 0 10 22
    13 0 15 0 15 0 16 0 0 19 0 0 20 21 0

                                        0 11 14 0 12 0 13 16 0


      ⎕←OBTSUC PREDEC
3 0 3 0 4 0 5 0 6 0 4 7 0 5 8 0 9 0 7 10 0 8 11 0
    17 0 14 18 0 0 0 20 0 22 0 22 0 11 0

                                        12 0 13 0 14 15 0 12 0 16
```

```
      ⎕←ADJAC←CBTADJ SUCCES
3  0  3  0  1  2  4  0  3  5  6  0  4  6  7  0  4  5  7  0  5  6  8  9  0  7  9  10
      11  0  10  12  22  0  11  13  14  0  12  14  15  0  12  13  16  0  13  16  17
      16  0  20  0  19  22  0  22  0  11  20  21  0
                                                                  0  7  8  10  0  8  9
                                                                  0  14  15  18  0  15  0

      SUCCES ADJA 11
10  12  22

      SUCCES ADJA 1
3

      SUCCES ASCEND 22
20  21  19

      SUCCES ASCEND 3
1  2

      SUCCES DESCEN 12
13  14  15  12  16  17  18

      SUCCES DESCEN 6
9  7  10  5  8  11  6  12  4  13  14  15  16  17  18

      SUCCES COFCO 3
3

      SUCCES COFCO 7
5  8  6  9  4  7  10

      SUCCES COFCO 10
8  9  7  10  5  6  4
```

```
DECOMP ADJAC

                         ****************************************
                         *  ELEMENTS REMARQUABLES DU GRAPHE    *
                         ****************************************

ARBORESCENCE : 1   3   2
ARBORESCENCE : 17
ARBORESCENCE : 18
ARBORESCENCE : 19   20   22   21
CYCLE : 4   5   6
CYCLE : 5   6   7
CYCLE : 7   8   9
CYCLE : 8   9   10
CYCLE : 12   13   14
CYCLE : 13   14   16   15
LOBE : 4   5   6   7
LOBE : 7   8   9   10
LOBE : 12   13   14   16   15

          ****************************************************
          *  ENCHAINEMENT DES ELEMENTS REMARQUABLES DU GRAPHE  *
          ****************************************************

LE POINT D'ARTICULATION 7 RELIE LE LOBE 4   5   6   7 AU LOBE 7   8   9   10
LE POINT D'ARTICULATION 4 RELIE LE LOBE 4   5   6   7 A L'ARBORESCENCE 1   3   2
LE POINT D'ARTICULATION 15 RELIE LE LOBE 12   13   14   16   15 A L'ARBORESCENCE 17
LE POINT D'ARTICULATION 16 RELIE LE LOBE 12   13   14   16   15 A L'ARBORESCENCE 18
LE POINT D'ARTICULATION 11 RELIE LE PONT 10   11   12 A L'ARBORESCENCE 19   20   22   21
LE PONT 10   11   12 RELIE LE LOBE 7   8   9   10 AU LOBE 12   13   14   16   15
```

MANAGEMENT OF APL TIME-SHARING ACTIVITIES

J. Higgins and A. Kellerman
Computer Center
State University of New York (SUNY)
Binghamton, New York

Introduction

The management of a terminal system at a university or industrial installation provides a formidable task. The user needs take arious forms: (a) an educational program in the syntax of the languge and techniques of programming to take advantage of the attributes of the language, (b) consultation on programming problems (both trivial requests and those involving desing, format, and construction of complex tasks, (c) publicity on operational considerations such as hours of operation, the location and availability of terminals, scheduling, etc., (d) documentation on existing programs and packages, and (e) assistance in administrative activities such as the restoration of working copies of damaged programs, groups, workspaces, etc., and the transportation of packages from our installation to another. The successful management of a terminal system such as APL then involves not only the proper maintenance and honing of the system to insure optimal utilization of computer resources for day by day activities, but well defined procedures for providing the additional personnel support to satisfy the above stated needs.

SUNY-Binghamton's APL System

SUNY-Binghamton has offered APL since the summer of 1967 and currently operates the XM6 version of OS APL/360. There are 1750 APL account numbers on the system; some of these shared by a number of users. Practically every department on campus - from theatre to geology and business to nursing - uses APL, with various emphases. In addition to supporting the local campus, seven sister SUNY institutions and six area high schools access our APL system.

The instruction and education section of the Computer Center offers a variety of forms of education and consultation to all users. Both potential and experienced users - students, faculty, and staff - feel free to request support at various levels and have received a reasonable degree of satisfaction. Initially, here as elsewhere, potential users were being solicited. Now, rather than acting as apostles and missionaries, we are in a position of resoponding to ever-growing demands for service from existing and potential users. This change in the nature of support is very gratifying, yet presents certiain problems.

Coincident with these rewards of satisfaction, the generation of enthusiasm, and staff motivation are the assorted and varied problems of developing the most effective system for all levels of user education, of effectively motivating and supporting worthwhile classroom and individual projects, of developing ways and eans of evaluating user and system performance, of maintaining the system, and of general administration with limited staff, facilities, and budget. These problems with these restraints are present to some degree in all installations supporting terminal systems. It therefore seems appropriate to present some of our experience, problems, solutions and attempts at solutions with the hope of developing a dialogue with other installations.

It is the purpose of this paper, then, to discuss those problems inherent in maintaining the system and in providing sufficient documentation and "publicity" on the availability of the system and its features, and in posing some partial solutions for providing support for ensuing generations of a core of competent and satisfied users.

Some Approaches

Terminal Allocation. It is an axiom of time sharing that accessibility of terminals to users increases usage to a very large extent. It is therefore desirable to have terminals dispersed in strategically located positions to encourage usage. This provides considerable problems, however. It is desirable to have terminals proctored for various reasons, including programming assistance, terminal maintenance, and general supervision of scheduling and use. These proctors are usually undergraduate students who, in addition to carrying out the above tasks, interface well with students and faculty on a one-to-one basis. Financially it is not possible to proctor locations, which are scattered around campus and house only two or three terminals.

For the most part we have avoided any serious problems by having some diverse locations of terminals periodically checked; two large terminal rooms, containing 22 terminals, are constantly proctored.

<u>System Maintenance</u>. In the spring of last year, 1971, space was rapidly depleting for saving of APL workspaces on two 2314 packs, with no prospect of adding an additional pack. APL users were encouraged to cut down on the amount of material saved and the additional workspace allotment was strictly controlled. However, by the beginning of May, with users getting "NO SPACE" messages, the situation was critical.

The policy at that time of deleting users who have been inactive for three months was not generating space fast enough.

As a last resort, all users were required to turn in a written form, giving their account number and the workspaces that they wanted to be maintained on the system.

After several abortive attempts resulting from misinterpretation of the documentation on the standrard APL IBM utility, lack of complete documentation on the APL utility, our inability to fool the utility into accepting an incremental dump tape for a full dump tape, and various other blunders, the following procedure was implemented. With the help of three programmers and a keypunch operator, cards were punched for each requested workspace. A full dump tape was made and new APL packs were created. From the APL utility, an ACCT 0 to tape was made. This ACCT 0 lists all users and account numbers. This tape was accepted with a program that punched cards with the system command)ADD for each account number on the old system. This deck was read into the 1050 (a terminal equipped with a 1056 card reader) to add all users to build the directories, to the newly created APL packs. The process was very slow, since the 1050 reads a card every 6.7 seconds and there were 1576 cards.

The worskspaces, for which "save forms" were turned in and for which a card had been keypunched, were restored to the new system, using the APL utility which can restore 100 workspaces at a time.

A complete backup set of tapes was kept for people who neglected for various reasons to submit save forms - for later recovery.

Although this method worked sufficiently well, there were several objections to it. First, the very fact that the final procedure was the result of a series of blunders with no better solution in sight left ilittle control. Second, the process of keypunching the cards for the workspaces and processing them through the 1056 reader was very time consuming. The paperwork was a nuisance. The retrieval of WS's from the old packs and the creation of the new packs monopolized the computer for a full day.

These disadvantages coupled with the fact that there was a general displeasure in the amount of information conveyed by the form of the APL account number lead to our present method. This method hopefully will be updated to something better, perhaps tied to the addition of files, to improvements in the APL utility, and to broadening the scope of the system commands to handle multiple entries.

<u>Account Numbers</u>. Instead of assigning numbers according to a 5 digit department code with the last four digits of a social security number, for a nine digit number, the following scheme was adopted. The first digit represents status, the next five represent department, and the last three are assigned sequentially according to edepartment. The status digit consists of 1-4 for undergraduates, 7 for graduates, 8 for faculty, and 0 for people who we feel need their number for only one semester. In the past everyone was arbitrarily assigned 1 workspace. Now 0 workspace quota is given to people who are using the CAI packages. The 0 numbers are deleted every semester; the 7 and senior numbers in June. The process for deletion is carried out in the following manner. A general purpose selection program is run against the APL ACCT 0 (under TSO) produced by the utility to search for 0's or 7's or any particular combination desired. This program creates a data set with the selected records. This data set is accessed with another program that punches cards with the appropriate APL system command and user number. These cards are processed through the 1050. The general purpose selection program can also select records of users of such combinations as all senior biology majors going to Corning Community College who have been connected to the system for over 35 hours. A developing interest in the school is the psychology of the time-sharing user. This type of output, along with information obtained from the I-beam readings, provide much information for on-line collection and analysis of data in this realm.

A similar procedure was used, that is punching a deck with the system command ")LOCK under number," to change over to the new numbering scheme. Users were given one month to copy old information into their new number. The old numbers were deleted with a ")DELETE" deck. We have a ")CONTINUE" deck that can periodically be read through the 1050 to clean up the CONTINUE workspaces that users fail to drop. (Figure 2).

SUMMARY OF THE DIFFERENT SPACE SAVING PROCEDURES

Before			After		
Users on System	Tracks	Workspaces	Users on System	Tracks	Workspaces
GRAND RETRIEVE (MAY)					
1748	7399		1748	4850	1576
LOCK-REASSIGN (SEPTEMBER)					
1640	6268	2010	1476	5395	1694
DELETE 0's STATUS NUMBERS (JANUARY, 1972)					
1769	7521	2461	1292	6572	2115
MARCH STATUS (CURRENT)			AFTER DELETING CONTINUES (MAY 9)		
1777	9025	3124	1777	6960	2288

<u>Use of I-Beams in Monitoring System Usage</u>. Using an APL function, MONITOR, requiring a privileged terminal, information can be obtained on a continuous basis on specific port usage, specific account number usage (such as histograms of graduate student usage throughout the day), and total numbers of users in a given time interval. From the data collected and from the results of any desired additional statistical analysis, decisions can be made concerning terminal usage, location suitability, suitability of APL schedule as well as information on amounts of use by different types of users. (Figure 3).

By using I-beams 1-14, which require a privileged terminal, various information about the APL system performance can be collected on line. (Figure 4).

The I-beams, representing histogram data, return a vector of integers seach element of which represents a full word of data. Since this information, collected from the time APL starts running until shutdown, is collected in half-word counters, each I-beam vector has to be decoded, split into its two half-word components with the following APL function. (Figure 5).

For example I2 - represents the system reaction time from when the user's return is detected, until his workspace is dispatched. (Figure 6).

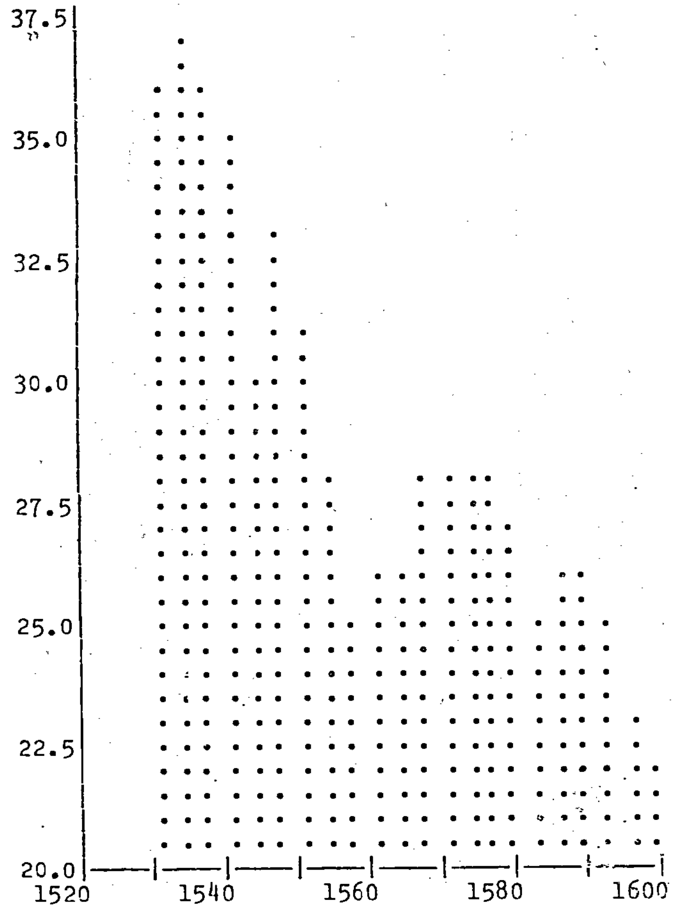
Although our experimentation with the I-beams is still rudimentary, attempts are being made to use this data as input to a simulated time-sharing system, for studying system performance under different loads, and for analyzing the behavior of the time-sharing user.

<u>Priority and Quantum</u>. When operating in a multi-programming environment, the effect - depending on factors such as configuration, number of terminals connected, types of jobs - of APL on batch jobs and vice versa can be substantial. Various parameters internal to APL can be adjusted. APL ensures that other partitions receive frequent CPU service by alternating its own priority between high and low. When APL has a low priority, other partitions will get CPU service. The normal proportion of time that APL has high priority is controlled by a function PRIORITY a,b, which is distributed with the workspace, OPFNS. The priority proportion varies approximately linearly from ------------ depending on the number of ports in use. There are other factors involved such as the quantum, the time allotted an active workspace in core, that can also be set internal to APL by an APL function.

Psychologically a time-sharing user desires at most a 3-5 second response time, (depending on the complexity of the request) but batch users object to at times 400% degradation in their jobs caused by APL. Hence some compromises have to be made. See Figure 7 for comparison data where the priority and quantum have been varied.

<u>Security of the APL System</u>. Theft of numbers of unauthorized users who search wastepaper baskets, unauthorized copies of OPFNS, disastrous experimentation by inquisitive but well-meaning users who desire to probe the mysterious inner workings of APL, and mischievousness make security an annoying but necessary task. As far as the Computer Center has determined no simple procedures or solutions are in evidence in the current IBM APL release. Various attempts at devising elaborate check functions for privileging "authorized" users at terminals outside the confines of the Computer Center have always been cracked.

Beyond these basic considerations there are the very real problems of offering security of creativity to those who desire it. With the possibility of patents for original algorithms and use of functions for trading with other installations or for publishing, workspace and function



NUMBER OF TERMINALS CONNECTED

VS

TIME OF DAY

Figure 3-a

NUMBER OF HOURS CONNECT TIME VS. PORT NUMBER.

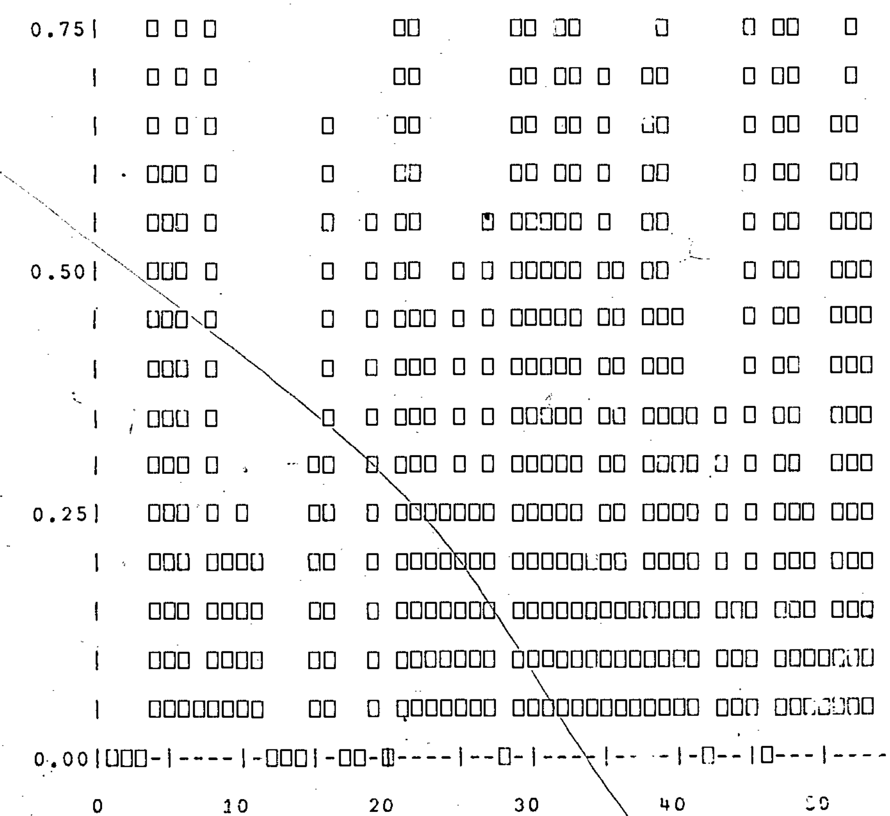


Figure 3-b

Figure 3-c

ACCOUNT NUMBER: 41001

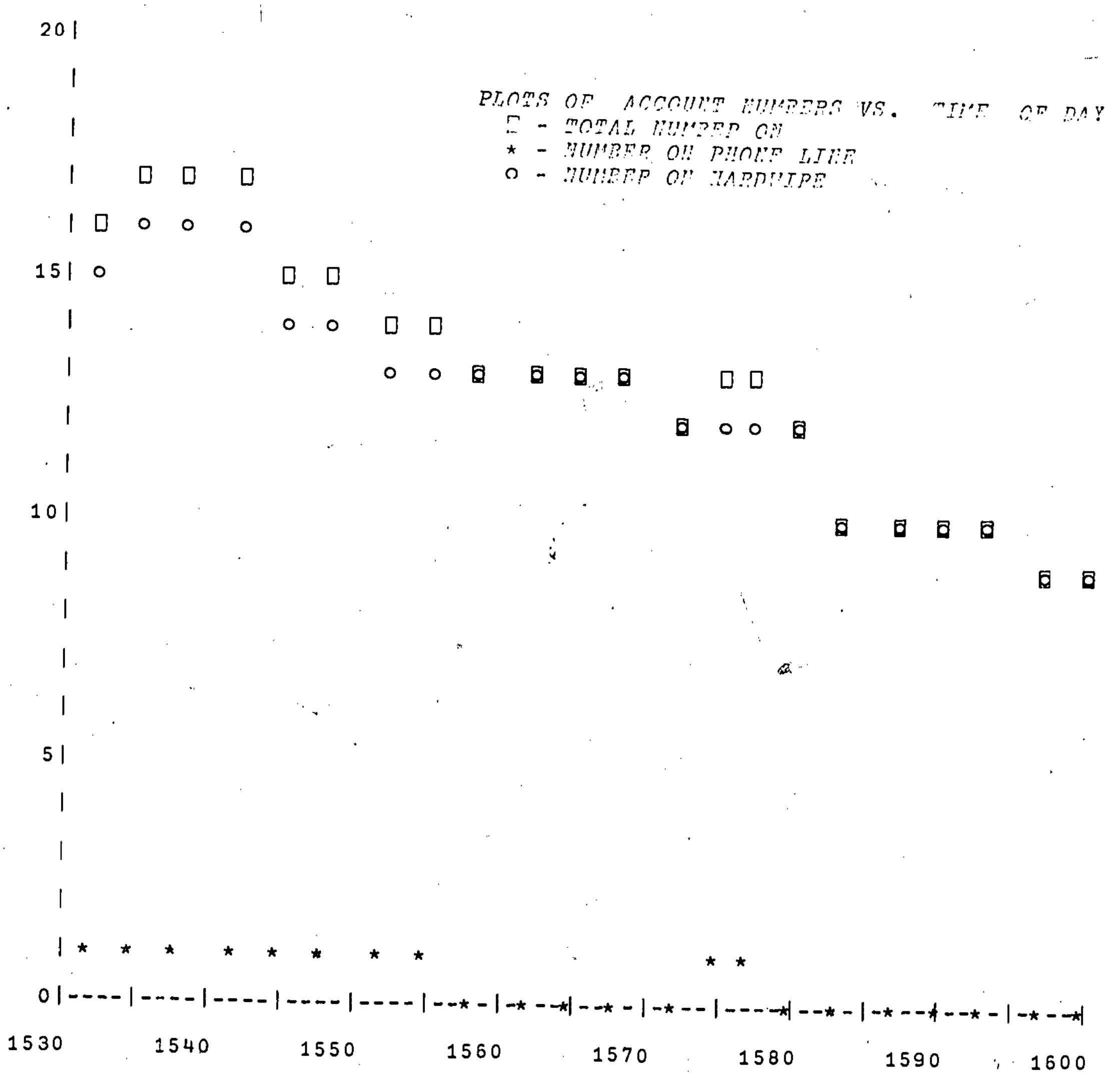


Figure 3-d

```
      ∇MONITOR[⎕]∇
    ∇ DELAYΔ MONITOR UNTIL;P;A;CO
[1]   TIMER←TERMNUM←⍳0
[2]   UNTIL← 72 60 ⊥2↑UNTIL
[3]   'INPUT ACCOUNT NUMBERS YOU WISH MONITORED ONE AT A TIME  INPUT STOP
[4]   NUM← 0 5 ρ0
[5]  L2:→(∧/'STOP'=4↑P←⍞)/EON
[6]   NUM←NUM,[1] 1 5 ρ 5↑P
[7]   →L2
[8]  EON:CC[illegible]←52ρ0
     [illegible]UM), 0 3)ρ0
     [illegible]OP:[illegible] DELAYΔ
[11]  TIME← 24 60 ⊥2↑ 72 60 60 60 ⊤I20
[12]  TERMNUM←TERMNUM,I23
[13]  TIMER←TIMER,100×+/(72 60 ⊤TIME)÷ 1 60
[14]  CONNECT←CONNECT+((⍳52)∊ON)
[15]  A←((1↑ρNUM), 1 3)ρCO←0
[16] INL:A[CO;1;]←(ρP),(+/P<21),+/21≤P←ACCT NUM[CO←CO+1;]
[17]  →(CO<1↑ρNUM)/INL
[18]  MAT←MAT,[2] A
[19]  →(TIME<UNTIL)/LOOP
[20] ABORT:'NUMBER OF TERMINALS CONNECTED VS. TIME OF DAY.'
[21]  30 90 PLOTT TERMNUM VS TIMER
[22]  10 1 ρ' '
[23]  CONNECT←(CONNECT×DELAYΔ)÷3600
[24]  'NUMBER OF HOURS CONNECT TIME VS. PORT NUMBER.
[25]  PORT←⍳52
[26]  30 90 PLOTT CONNECT VS PORT
[27]  CO←0
[28]  10 1 ρ' '
[29]  ' PLOTS OF  ACCOUNT NUMBERS VS.  TIME  OF DAY'
[30]  '    ';PC[1];' - TOTAL NUMBER ON'
[31]  '    ';PC[2];' - NUMBER ON PHONE LINE'
[32]  '    ';PC[3];' - NUMBER ON HARDWIRE'
[33]  3 1 ρ' '
[34] LP:' ACCOUNT NUMBER: ';(' '≠NUM[CO;])/NUM[CO←CO+1;]
[35]  →(0≠+/+/+/MAT[CO;;])/(I26)+2
[36]  →((I26)+2),0ρ⎕←'NONE OF THIS ACCOUNT NUMBER SIGNED ON'
[37]  30 90 PLOTT MAT[CO;;] VS TIMER
[38]  '


[39]  →(CO<1↑ρNUM)/LP
    ∇

      ∇ACCT[⎕]∇
    ∇ R←ACCT N;OR;X
[1]   →((∧/'0'=N),1=ρN←(' '≠N)/N)/ZER,STAT,0ρOR←6I 0 0
[2]   R←1↓⌊(7I108)÷1000
[3]   R←(10*5)×R-⌊R←R÷10*5
[4]   R←⌊R÷10*5-ρN
[5]   →ZER-2
[6]  STAT:R←1↓⌊(7I108)÷10*8
[7]   R←(X∊R←(R=10⊥'0123456789'⍳N)/ALL)/X←ON
[8]   →ZER+1
[9]  ZER:R←(((1↓7I108)>1000000)∧(1↓7I108)<100000000)/ALL
[10]  OR←6I0,OR
    ∇
```

Note: Good for 52 ports. Additional functions used are found in the Operator's workspace.

Figure 4

I-Beam Number	Unit	No. Of Elements	Max. Value	Significance
0	-	13	-	Count of special disk operations. The elements of the decoded vector give the number of times each of the following system commands has been used: DROP, SAVE, LOAD, COPY, ADD, LIB, OFF, DELETE, LOCK, UNLOCK.
1	1 percent	100	100	The percent of elapsed time given to service an input.
2	1/60 sec.	240	4 sec.	The system reaction time from when the user's return is detected, until his workspace is dispatched.
3	1 second	120	2 min.	User keying time, from the time the keyboard unlocks, until the user hits <u>return</u>.
4	1/60 sec.	120	2 sec.	Compute time per input.
5	(a transfer vector of absolute addresses)			
6	1 minute	120	2 hours	Connect time for each session.
7	1 second	240	24 sec.	CPU time for each session.
8	1 byte	148	148 bytes	Raw input character count, including backspaces, etc.
9	1 second	120	2 minutes	Input arrival time (from one carrier return to the next).
10	1 byte	148	148 bytes	Internal output line length.
13	250 bytes	200	50000 bytes	Garbage in workspace at time of swap write.
14	250 bytes	200	50000 bytes	Active size of workspace at time of swap write.

Figure 5

Decoding the I-Beams

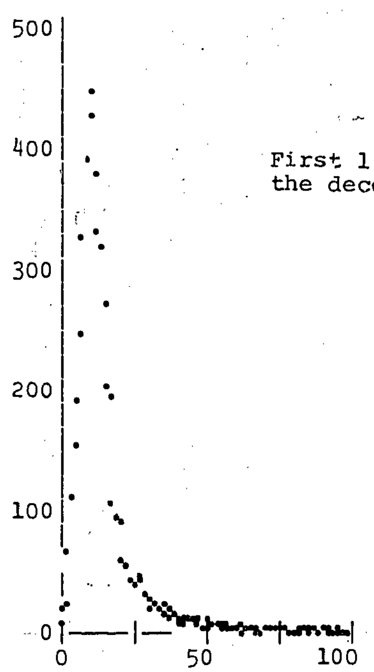
The data in core storage is read-out by a histogram I-beam in increments of full words. To decode the I-beams the following APL function can be used.

```
      ∇SPLIT[⎕]∇
    ∇ R←SPLIT I
[1]   R←(1↑I),,⍉ 65536 65536 ⊤1↓I
    ∇
```

Figure 6

```
       IBEAM
THIS WORKSPACE HAS COLLECTED A GRAND TOTAL OF 1 IBEAM READINGS.
THEY ARE FOR THE FOLLOWING DATES AND TIMES:
01)   10:49:19AM ON 02/14/72
WHICH DATE AND TIME DO YOU WISH? (SPECIFY BY NUMBER)
⎕:
       1
WHICH IBEAM  (0-14 EXCEPT 5) DO YOU WANT?
⎕:
       2
DO YOU WANT THE UNCODED IBEAM DISPLAYED?
⎕:
       YES
371202  31064072  1310742  4456558  10158272  16318793  25559486  27984253  21627201  17694924  12845163  6226010
       3997752  3538989  2621478  2818094  1966109  1245207  1441812  1507343  1245197  1048589  458762  458765
       655372  655367  262148  720903  458758  458756  458757  262148  327680  65540  262146  65540  65541  196610
       131074  196611  196611  1  65538  65540  65538  131076  65540  262145  196608  262144  1  65537  0  65536
       131072  65536  196609  196612  65538  65539  65536  65537  3  4  0  196609  0  0  0  2  1  0  3  65536
       2  65536  1  196608  65537  2  131072  65536  65536  131073  65536  1  65536  2  65536  131072  0  0
       0  131072  2  0  65536  65536  65536  65536  65537  0  65537  1  0  1  65538  1  0  2  2  1  0  0  65536
       131072  0  0  0  0  131072  184

  DECODED VECTOR IS:474  8  20  22  68  110  155  192  249  329  390  446  427  381  330  321  270  204  196
       107  95  90  61  56  54  45  40  38  43  46  30  29  19  23  22  20  23  15  19  13  16  13  7  10  7
       13  10  12  10  7  4  4  11  7  7  6  7  4  7  5  4  4  5  8  1  4  4  2  1  4  1  5  3  2  2  2  3  3
       3  3  0  1  1  2  1  4  1  2  2  4  1  4  4  1  3  0  4  0  0  1  1  1  0  0  1  0  2  0  1  0  3  1
       3  4  1  2  1  3  1  0  1  1  0  3  0  4  0  0  3  1  0  0  0  0  0  0  2  0  1  0  0  0  3  1  0
       0  2  1  0  0  1  3  0  1  1  0  2  2  0  1  0  1  0  2  1  1  0  0  1  1  0  0  2  1  0  2  0  0  0
       0  0  0  0  2  0  0  2  0  0  1  0  1  0  1  0  1  0  1  1  0  0  1  1  0  1  0  0  0  1  1  2  0  1
       0  0  0  2  0  2  0  1  0  0  0  0  1  0  2  0  0  0  0  0  0  0  0  2  0  0  184
 A PLOT OF THIS IBEAM IS AS FOLLOWS:
THE NUMBER OF OVERFLOWS WAS: 184
THE TOTAL OFF-SCALE READINGS WERE: 474
```

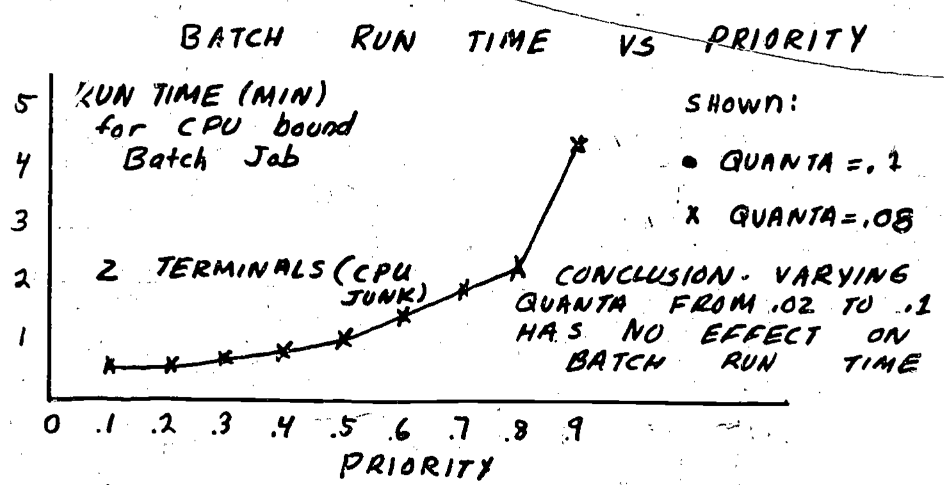
I2: The system reaction time from when the user's return is detected, until his workspace is dispatched.

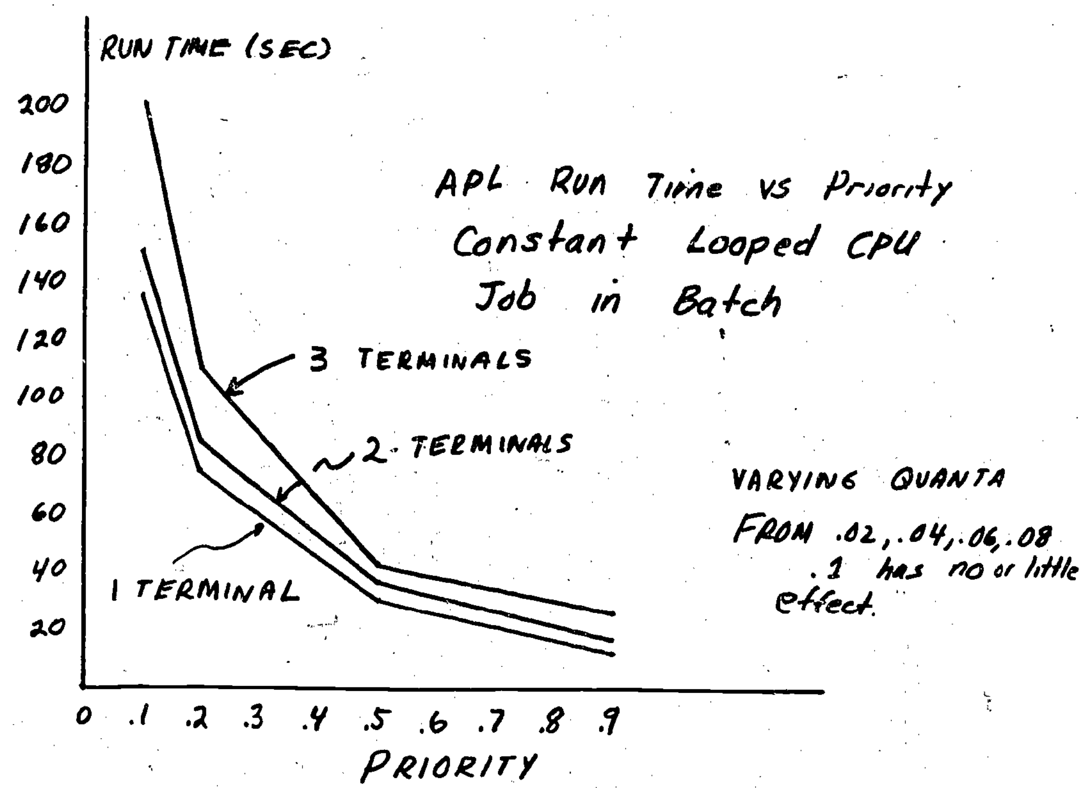


First 100 elements of the decoded I2.

Units of 1/60 of a second

Figure 7


BATCH RUN TIME VS PRIORITY
RUN TIME (MIN) for CPU bound Batch Job
5
4
3
2
1
0
2 TERMINALS (CPU JUNK)
SHOWN:
• QUANTA = .1
x QUANTA = .08
CONCLUSION - VARYING QUANTA FROM .02 TO .1 HAS NO EFFECT ON BATCH RUN TIME
.1 .2 .3 .4 .5 .6 .7 .8 .9
PRIORITY


RUN TIME (SEC)
200
180
160
140
120
100
80
60
40
20
0
APL Run Time vs Priority
Constant Looped CPU
Job in Batch
3 TERMINALS
2 TERMINALS
1 TERMINAL
VARYING QUANTA FROM .02, .04, .06, .08 .1 has no or little effect.
.1 .2 .3 .4 .5 .6 .7 .8 .9
PRIORITY

security and some form of credit has to be given to programmers and researchers while encouraging APL users to share their work with the world.

Another security problem of a different nature is the disappearance of the contents of workspaces accidentally or because of internal damage. If users report their mishap to the Center within the period of a cycle of dump tapes (3 weeks) their workspace can usually be recovered. We do not publicize this capability.

Occasionally workspaces come out DAMAGED (IMPEACHED) in the twice weekly dump-restore procedure meaning that they are questionable, but are dumped and restored. We try to eliminate the questionability by either copying them over or restoring from a backup tape. Whether this is necessary or not, we do not know. For example, we have discovered that the upper and lowercase idle character cause an impeached workspace, but use of the functions containing these characters is in no way hampered.

To our dismay in March, because of later to be discovered problems in formatting the disks, we had several workspaces DAMAGED and REJECTED as opposed to being DAMAGED and IMPEACHED. The contents of these workspaces are gone. But their name remains in the directory. If a user attempts to save into these workspaces, APL abends. It is necessary to bring APL up again and at all possible speed)DROP these workspaces from the directories.

Education. There is no undergraduate computer science program at SUNY-Binghamton, hence there are no "programming" courses. There are graduate courses (in APL & PL/1) in the School of Advanced Technology which undergraduates can take, by petition, for credit. The vast majority of students, staff and faculty look to the Computer Center for instruction. We have produced a series of video instruction (9 tapes, 45 minutes each) which introduce students to APL/360. The tapes are run, with a knowledgeable Center employee in attendance, twice a semester - usually twice a day (noon and 6:00 p.m.). From two to three hundred people per year are introduced to APL in this manner. The Center employee is necessary primarily to provide the encouragement and motivation for all students to get on the terminal as soon as possible and not just take the video course for theory. During each summer a special class for faculty only is held, and is well attended. There is a 50 page supplementary manual available which is used to follow the video tapes - primarily for the purpose of discouraging note-taking during the tapes. Copies are available.

In addition to the video classes, which are well received, we offer live classes for groups, such as individual classes, on a demand basis. There is also a series of workspaces in APL which instruct a user in the APL syntax in a "CAI" mode. We offer personalized instruction to users who read the User's Manual or our Quick Guide on their own. The Quick Guide is a brief introduction to APL/360 with notes concerning specifically our installation, and sample executions of some of our public libraries. This guide was written to hand to people who stop in the day after the APL classes end and ask when we'll be teaching an APL class. Copies are available on request.

Assistance to Users

In providing assistance to faculty we find various categories of needed support: (1) those who are sophisticated programmers, have good ideas for applications in their courses, and merely request account numbers for their students, reservation of terminals, and perhaps demonstrations of APL in group sessions; (2) those with good plans for applications, but lack ideas for implementing them; (3) and those whose only attribute is enthusiasm. The last two categories of people are best handled on a one-to-one basis, trying to adapt their needs to techniques and existing programs. With a sufficient number of examples of problem-solving techniques, simulation and tutorial programs they can find something consonant with their interests that will provide the supplement or embellishment to their course that they sought.

Students

Students who use the APL system do so also for various reasons: (1) course requirements, (2) their own research or other class work, (3) general unguided curiosity, and mass productions of SNOOPY posters. However, it is true that most students become, for various reasons, much more sophisticated and elegant APL programmers than faculty and that a great deal of course-related APL work can be traced to student initiation by suggestion or actual development.

Computer Center Assistance

In most cases, APL projects have the most success when they are tailored to a specific professor and class. We have, however, developed some general purpose CAI techniques that are applicable to various circumstances. We are currently evaluting the Author Tutorial System

available through IBM. The object is to allow professors, with a limited knowledge of APL, to construct tutorials and drills. Students can respond to questions in free form sentences. Statistics of student performance can be obtained.

In terms of particular applications, some ideas have required a great deal of effort on our part and on the part of the originator. The first step is to determine whether or not the project is "worthy" of implementation.

The determination of the "worthiness" of a given application is not well-defined. Probable use, time, and limited personnel constitute the primary constraints. It frequently goes beyond differentiating what is or is not a good APL application. What we may conceive of as a "bad" application can, in some instances, serve a definite need. Many applications such as the CHEMLAB, APL laboratory monitor, are not cost effective yet, but represent excellent prototypes.

Certain modificaitons of ideas and procedures invariably must occur to make them suitable for APL implementation. Interestingly enough, because of the ability of APL in simulating experiments to their very limit, many of the planned Freshman Physics lab experiments were modified - mainly because the original experimental procedures, taken to the limit, produced less accurate results than the modified procedures on APL. The necessary algorithms must then be developed, and then coded. Program editing, a continual dialogue between programmer and originator, is an interactive process that can be very time consuming. Once the programming is completded, arrangements are made to allow students easy access to the programs. Student reaction is an important ingredient in the determination of modifications and embellishments.

Documentation

It is axiomatic in user service-oriented organizations that effective publicity is an all-important ingredient for success. We publish the Computer Center Newsletter four or five times a year. (If you'd like to be on our mailing list, we have applications with us.) APL news gets the most coverage. We also publish a list of public library workspaces and their contents; we also have copies here for distribution. We also maintain standardized "on-line" documentaiton.

A large number of our APL users are interested in statistical functions. We have two statistical packages: STATPAK and a package from New Paltz. Several additions have been written by SUNY-Binghamton people. Unfortunately, a large portion of potential users know statistics but cannot understand the descriptions that use a large amount of APL terminology. Our student proctors know APL, but not statistics. We have developed a descriptive workspace STATHELP which gives even additional help to bridge this gap.

We have implemented a MATRIXHELP that describes some things that can be done with matrices and APL and points to other matrix workspaces in the public libraries. A FORMATHELP workspace contains functions and help to format data, functions and help in writing CAI and directions on use of the various plot functions that have accumulated in our libraries.

Future Efforts

Some areas of future emphasis, in addition to those mentioned above, include more concentration in Psychology, the School of Management, the School of Nursing, and applied mathematics in the School of Advanced Technology (SAT).

Since we do not currently have a file system with our APL system we are restrained by the 36K WS limitation - especially for statistical applications, long simulations, and CAI programs requiring the logging of student statistics. We feel a distinct need to provide more information to users on good programming habits and on time/space tradeoffs. A programmer in SAT, Grant Sullivan, has done some investigation in Programming techniques to save space and time/space tradeoffs. His work provides some help and guidance in good programming techniques in the above areas.

Of the 1750 users on our system a relatively small proportion exhibit exceptional programming skills. It is a testimony to the efficiency of the APL/360 implmeentation that less than good programming does not necessarily punish the user. There are users who are very clever with the APL syntax but do not use it well in everyday practice. There are, of course, users who, no matter what the amount of effort, will never be good programmers. It is probalby impossible and certainly impractical to impose restrictions on the user community to attempt to enforce programming standards. We would, however, like to increase computer-related skills in all areas.

Some of this improvement comes with knowledge of basic computer concepts and numerical methods. Most of our users do not have the time to dedicate several courses to achieve this type

of knowledge. So we [illegible] in the poosition of having to consider ways of capsulizing APL and statistics, advanced APL, numerical methods useful in coding APL problems, etc.

Finally, there is the task advising users what system to use. Initially we did not offer any choice of conversational terminal facilities. However, we currently run TSO and anticipate situations, such as taking the determinant of 50 x 50 matrices, where our advice will be to channel the application to the most applicable terminal system.

The ultimate goals are to transport as many useful programs to our system as we can, to provide a large base of available routines, to encourage the development of curriculum materials in our consortium, to adequately publicize that which is available, and to provide the consultation and assistance necessary to eiliminate or reduce impediments to general development. We feel we are in an embryonic stage now, but look forward to increased service to our users. We would appreciate sharing experiences and programs with other installations. There is much to be gained by cooperative efforts.

EVERY LITTLE BIT HURTS:

Saving Money by Saving Space in APL

Richard Alercia, State University of New York
Robert Swiatek, Binghamton Public Schools
Gerlad M. Weinberg, State University of New York
Binghamton, New York

The State University of New York at Binghamton has been a user of IBM's APL system since the earliest releases, and currently runs about 50 ports on a model 155. At the time of our study, 1400 user numbers accounted for approximately 5000 tracks of 2314 space. A persistent trouble in our system seems to be one of inadequate space on disk files. At the beginning of a semester there is quite a bit of space, but as the weeks pass on, the disks fill monotonically until users begin to feel the effects. Each passing year seemed to see the addition of another 2314 to the system, in order to solve the previous year's problems, and each year the system fills up. Other installations have told us of the same difficulty, so we decided to investigate the problem.

We conducted our investigation during the Fall of 1971, through a survey and through detailed investigation of a random sample of individual users' workspaces. Our sample generated 295 user numbers, two of which were no longer in use and 48 of which were locked. We respected all locked numbers and workspaces--of which there were only seven under unlocked numbers. Altogether, then, we investigated 245 numbers. We realize that the profile of locked numbers may be different from that of the unlocked, but we do not know how to adjust for the difference. We conjecture that the average locked user is a larger, more sophisticated user, and since they seem to waste more space among the unlocked, we imagine that our esimates are consequently on the conservative side.

In parallel with the study of workspaces, we distributed a questionnaire to 35 persons, most of whom were in the School of Advanced Technology and so might be expected to be more knowledgeable about APL than the average user at Binghamton. The purpose of the questionnaire was to estimate what effect knowledge of APL has on space usage, and what possibilities there might be for space savings. We also carried on a number of informal discussions with users, and observed users at work.

State Indicators

One of the first sources of wasted space is the space used for suspended functions. If a function is suspended during execution and the state indicator is not cleared, a certain number of bytes gets wasted unless execution is resumed. Of the 245 users we checked, 48 had some such wasted space. The number of bytes ranged from 68 to 20, 504 per 32K workspace. The total for the sample was 75,994, or about 300K bytes per 1000 users. For our whole installation, if this rate is representative, 420K bytes are consumed in this manner, or about 70 tracks.

Our survey showed that at least one-third of the respondents did not know the meaning of the SI symbol, and recall that this survey was among the most knowledgeable group of APL users on campus. But even people who knew what SI meant had wasted bytes in their workspaces and had bytes consumed by SIs, even damaged SIs. Indeed, the more "sophisticated" the user, the more space he wasted in this way.

Duplicate Workspaces

We found a number of people to have two workspaces which were precisely the same, while others had very similar workspaces. We also found two or more users who had identical worskpaces. The main reason for such duplication is the presence of a CONTINUE workspace, which is presumably around in case of, or because of, system difficulties.

In our sample, we found 92 CONTINUE workspaces for 245 users. Erasing only the ones which were exact duplicates of other workspaces would have reduced the 640 tracks used by these 245 users to 509 tracks. Dropping all CONTINUE workspaces would have reduced the total space to 363 tracks. This saving comes to approximately 1130 tracks per 1000 users, or one 2314 for between 3,000 and 4,000 users.

Handling of Libraries

Of major importance is the space wasted by copies of functions which can be found in some other workspace. These could be in either the APL public library or some private library. While it is difficult in the present APL implementation to say precisely, our best estimate of the number of public library functions per workspace is two. The average size of these functions is approximately 2000 bytes, or 4000 bytes per workspace. Over the 1657 non-CONTINUE workspaces in our system, this represents about 6,600,000 bytes, or more than 20 percent of the total tracks. If CONTINUE workspaces are included, this duplication of public functions accounts for over one-quarter of the space consumed in our system.

Space consumed by non-public, or semi-public, functions is difficult to estimate, but we found several sets of functions which were duplicated quite frequently. Most of these seem to have originated in some teacher's workspace. A typical situation is for an instructor to create functions for his class to use--functions hich each member of the class saves in his own workspace. One case, for instance, involved two tracks, so if there were 20 people in the class, 40 unnecessary tracks would have been consumed.

Miscellaneous Wastage

Disk space also gets wasted in numerous small ways which will probably be inaccessible to any systems solution. A person who knows the system well potentially has good control over his space utilization. Given an incentive to save space, he will be able to do so. On the other hand, a person using the system with little knowledge will most likely be wasting space, even if he has reason not to. Our survey indicated that many users were not sufficiently aware of the workings of the system to save space even if they had wanted to.

But knowledgeable users can also be space wasters. Though they know how, they are simply too lazy to clean up their workspaces - especially since our installation does nto charge for disk residence. We ran across one person who had seven workspaces, five of which were the same. This totals 166,000 wasted bytes, or roughly 24 tracks.

Some workspaces had functions which were obviously to be used only once, yet they were saved. Others had two or more similar copies of a function, one of which, at least, contained a syntactic error and thus couldn't possibly run. It is probably of no value to save a function which won't execute, especially if the number hadn't been used for six months.

Though we found no trace of it, there was a well-organized APL baseball league going around campus. The grapevine tells us that these workspaces are carefully locked. We did, however, find some unlocked workspaces with other games. One was a basketball workspace featuring the New York Knicks. Whether or not these functions had value for teaching or learning, we leave to others to decide.

We encountered one copy of APL NEWS OF THE WEEK from 1969. Another individual had a single function in his workspace which printed five other APL numbers. In additon, we found one unlocked MISS APL - using 7000 bytes - and three unlocked SNOOPIES.

Recommendations

There are a number of rather clear steps which could be taken by the IBM APL implementation to cut down on the amount of wasted space on disks. We shall consider these recommendations in turn, indicating which problem they address and how much saving they might be expected to realize in an installation such as ours which might be reasonably typical for a University environment.

State Indicator Vector

For 99.9 percent of the users 99.9 percent of the time, saving the state indicators of the program after an attention seems a waste of space, since few programmers know about their existence, fewer know their meaning, and almost nobody uses them to resume execution. Nevertheless, the advantages of saving the state can be maintained without the wastage of space if a few simple changes are made.

The system could prevent automatic storing of the state when)SAVE is executed. The state would only be saved when a special version of)SAVE is executed - such as

```
)SAVE WSNAME SI
```

This approach leaves only system shutdowns or crashes to contend with. In these cases, one may be interrupted unintentionally, so the state could be saved - but erased automatically after the first)LOAD or)COPY, and in any case after, say, one week. If the programmer hasn't loaded the interrupted space after one week, it can hardly be an urgent matter.

Estimated savings from this approach are 50 tracks/1000 users.

CONTINUE Workspaces

We recommend, at a minimum, that CONTINUE workspaces not be available except as an emergency storing place in case of sytem crash or shutdown. Many users employ CONTINUE to gain an effective increase in workspace quota, but they can be satisfied by whatever regular assignment procedure exists. In any case, it is poor practice to save in CONTINUE, for a sytem crash while working on something else will wipe out that version of CONTINUE.

In the case of emergency saving, CONTINUE workspaces should be retained for a maximum time of, say, one week and then automatically dropped from the system. Moreover, when a user signs on, he should be given a message that he has a CONTINUE workspace, and asked to use it right then or lose it. Unfortunately, this strategy is not sufficient, for the great majority of CONTINUE workspaces are simply sitting behind numbers which won't be used for months. Therefore, a time limit must also be set on inactive numbers.

In our system, implementing this policy would save 1130 tracks/1000 users. After our study, our Computing Center instituted the policy of periodically erasing all CONTINUE workspaces. There seem to have been no complaints, and the savings are commensurate with our estimates, thus proving an empirical demonstration that this approach works and is not unbearable to the users.

Handling of Libraries

Probably the greatest wastage of space in our APL system is brought about by duplicates upon duplicates of certain library functions stored under number after number, and sometimes many times under the same user number. When a workspace containing a locked function originally loaded from a public library is saved, only the name of the function need be saved. Then, when a)LOAD or)COPY is executed, the copy is brought anew from the public library. Currently, we estimate that this operation would save 20-25 percent of the disk space in our system, but this space saving would tend to grow as the library grows and the users stay longer with the system, so that their knowledge of the library grows.

This operation is almost transparent to the user, and would be entirely so if it were not for the possibility of new versions of library functions being issued from time to time. If the library functions are functionally equivalent, but are improved in space and/or time, this system has the further advantage of giving all users the benefit of the latest improved library routines. Only if functional changes are made could a user get into difficulty with a program not working which once worked. In any case, such troubles can be prevented by issuing the new version with a new name - which is probably best if the function has changed.

Can the User Do It?

An alternative solution to this same set of problems is to modify user behavior. Well-trained and conscientious users would, before saving any workspace, clear the SI and generally clean up garbage in the workspace. They would certainly not store copies of library functions, but would erase them before saving and copy them at their next work session. When loading after a crash they would carefully drop CONTINUE.

Were all our users like this, our APL system would be a neat and trim little operation. From our survey, however, we cannot find any evidence that any users are like this. While they may spend hours trimming a few bytes so as to make a job run in one workspace, they will not spend a few seconds copying library functions anew with each load. On the contrary, those users who do know enough to save space for the system will readily do the opposite if it is to their advantage.

For instance, at the peak of last year's space crisis, users began to experience NO SPACE messages when they tried to save their active workspace. In order to avoid losing any work, knowledgeable programmers filled each workspace with long vectors so as to ensure a full five tracks would be occupied. This prevented NO SPACE - for them - and they could shorten the dummy vector as needed.

At the other extreme, we find those users who might be happy to cooperate with the system, in saving space, but don't even know what "space" is.

Conclusions

No doubt an APL installation with one hundred percent knowledgeable and conscientious users could save much more disk space than the system changes recommended in this paper. No doubt, too, there will never be such an installation. Given the realities of user ignorance and selfishness for the large majority, significant dead storage savings must come from system changes - changes which are transparent to the user, or at worst within the override control of the knowledgeable programmer.

When APL was a young system, users could afford to put up with such glaring inefficiencies in storage management. In the first place, the number of users is always smaller when the system is first installed; in the second, the space per user grows as the number of users grows, so that the total space grows faster than linearly with time. Installations cannot go on indefinitely devoting additional disk packs to APL with each passing year. Charges for space can be expected to provide feedback to the users which will ultimately stabilize space per user, but once charges for space are instituted, users themselves will begin to edemand the kind of automatic space controls we have suggested.

In any case, these three simple systems changes we propose would reduce the configuration needed in our installation by one 2314 in three. Installations with larger numbers of users could expect proportionately larger savings in disk rental - rental which is effectively rental on an inefficient systems design.

ACKNOWLEDGEMENT

We should like to thank all the members of our Computing Center staff who gave us support and encouragement, but especially Ms. Anne Kellerman. We would also like to thank our 245 anonymous participants who (unknowingly) permitted us to examine their deepest secrets.

SECURITY OF APL APPLICATIONS PACKAGES

Paul Penfield, Jr.
17 Bradford Road,
Weston, Massachusetts 02193

By the term "applications package" is meant a set of interacting APL functions and variables that a user calls, along with certain "background" functions and data that are only called indirectly. If there is a proprietary interest in the package, then it is necessary to devise techniques to assure the security of the package.

An interpreted language like APL might be thought to pose severe security problems, since (unlike a compiled language such as FORTRAN) the source code is always somewhere nearby. However, with proper design, reasonable security (consistent with the value of the goods protected) can be achieved. This paper deals with security for packages installed both on private computers, and on commercial time-sharing systems, with the emphasis on the latter.

Why Security

The purpose of a security system is to make it more difficult (i.e., more expensive) for a potential thief to get at the package without authorization, than it would be for him to do so legally.

More specifically, there are four general tyes of acts that a security system should protect against. First is display of the functions (and possibly the data), for example by someone trying to discover the coding. Second is unauthorized propagation of the package, for example by means of DUMP's. Third is modification of the package, and fourth is unauthorized use.

There are four classes of people that the security system is directed to. First are unauthorized users. Second are authorized users, or users that are authorized for only certain types of use. Third is the operations staff of the computer, and fourth are computer system programmers. There is no effective way to prevent a system programmer, if he wishes, from violating the security of the package. I assume that any system programmer knows how to access the symbol table, how to unlock functions and how to beat the file system. The security of a package relies on the fact that such people are generally not dishonest, and therefore try to "play the game" fairly, and will not go out of their way to steal the package. It is therefore sufficient to design the system so that they do not stumble upon any secrets by accident. As for the operations staff, this is a larger set of people and it is probably wise not to tempt them. What is necessary is to design the package in such a way that nontrivial work is required for them to "beat" it. As for the end users, a few of them will regard it as a challenge to try to beat a security system, and therefore the security aspects must be designed as though all end users were malicious, scheming, knowledgeable APL programmers intent on stealing and/or destroying the package.

How Secure

No security system is foolproof. Fortunately, however, foolproof behavior is not required. The only requirement is that the cost of beating the system be, and appear to be, greater than the worth of the goods it protects.

In the case of an applications package, the worth has two separate upper bounds although if there is a data bank, the value of the data might be higher. Any package will by necessity have a user's manual which defines the interface between the user and the package. A competent APL programmer who is also knowledgeable in the particular discipline can, in principle, duplicate any package merely from the user's manual. One upper bound to the worth of a package is the cost of doing just that. The other upper bound is the price charged for the sale, lease, or use of the package.

Protecting Against Unauthorized Display

Unauthorized display of functions can of course be prevented by simply locking them. In some cases locking the workspace may also be a slight help, but in typical packages the workspaces in question are supposed to be available to users and it would not be appropriate to lock them.

Some installations have provisions for rendering functions and variable names unprintable. I have never been told, but I assume that this is done by changing the name in the symbol table to blanks, or to a "name" including a nonalphanumeric symbol or starting with a blank. This of

course should only be done to names that are not part of the user's vocabulary (i.e., only local variables and background functions and data) but is effective in preventing the display of some variables, especially local variables during program suspension.

Some installations have the ability to make a variable unprintable by changing its type, but if this can be done by the user it can also be undone and therefore is not an effective security measure.

Protecting Against Unauthorized Propagation

One way of propagating a package, of course, would be to display the functions and data and then manually re-enter them elsewhere. There are other methods, however. The most obvious is to request a selective dump of a workspace and then carry the magnetic tape to another installation. Some installations have conventions by which certain comments in a locked function make it un-dumpable, but aside from these there is little that a package designer can do to prevent a DUMP.

There is much that he can do, however, to make such propagation fruitless. Most installations have special individual features; the common ones are file systems, and fast formatters. If the package is individually tailored to use these features, it will not run on other systems.

There is some danger of unauthorized propagation by actual reproduction of the tape which originally carries the package to the installation. There are a couple of easy things that can be done to make this ineffective. First, there is the "you add the eggs" approach whereby a simple but necessary variable is omitted, and is then inserted manually after the package has been loaded. The other is to send the tape with one or more functions unlocked and in error. Then, since only you know what corrections are necessary, only you can make them, and the tape without the corrections is worthless.

Protecting Against Unauthorized Modification

With a package containing many functions and variables there is some danger of wrong results if some of the functions are replaced or missing. To protect against this, and therefore to preserve the integrity of the package, there is little that the package designer can do. What is logically required is the concept of a "locked group" which during *)COPY*, *)PCOPY*, *)GROUP*, and *)ERASE* commands, would always stay together. Another useful feature of such an arrangement would be identification of the locked group with the user number of the person who locked it, and therefore "owns" it. The rule then would be that only the owner could *)SAVE* a locked group; if anyone else tried to *)SAVE* a workspace containing the group, it would be erased before the save is executed.

However, this is just a suggestion. It is not implemented, and usually there is nothing a package designer can do to prevent modification of his package (although one installation has a similar arrangement applying to workspaces).

Protecting Against Unauthorized Use

Protecting against unauthorized use requires a validation system of some sort. Such a system need not be absolute, in the sense that it need not protect all functions in the package. It is sufficient to protect certain "critical" functions. This approach reduces the number of validation tests, and also eliminates many lines of code.

The cost of repeated validations can be eliminated by somehow "conditioning" the workspaces to allow subsequent use without another validation. This can be most easily done by setting a global variable, called the "conditioning variable," to match the results of a test calculation. If the conditioning variable is OK, computation proceeds; if it is not, the validation routine is called and either the user is validated and proceeds (without even realizing he was validated), or else the calculations are aborted. For this arrangement to be effective, the conditioning variables must not have an obvious value. The formula for calculating it must be secret, and new versions of the package should incorporate new formulas with new arbitrary constants.

A conditioned workspace must not appear conditioned under too general circumstances. For example, it would be bad if an authorized user could condition the workspace, *)SAVE* it, and then have an unauthorized user *)LOAD* it and proceed. Similarly, if it is ever desired to remove authorization from a user, the conditioning must be set so as to expire automatically. These conditions can be assured if the value of the conditioning variable depends upon I29 and I25.

A separate function to do the validation process should be avoided, since a user can substitute his own version. A better plan is to incorporate the validation code right in one of the critical functions of the package. If for some reason it is necessary to use a separate validation function, it should be written so that it works properly only within the environment provided by the functions that call it. This may be done by referencing local variables in the calling functions, or by references to I27.

Whether or not a separate validation function is used, a user can interrupt within the function and then branch to any line number. For this reason it may be wise to incorporate an I19 check to foil such attempts.

If possible, the conditioning variable and the validation routine name should be unprintable.

If the list of authorized users is kept in the workspace, it takes up space, so perhaps the validating algorithm should reset it to a scalar. On the other hand, if there is a file system, the list of valid users can be stored in a read-only file, with the user numbers coded in some way. The file password (if one is used) should be secret, and should not appear as part of a variable which can be displayed during program suspension. The list of authorized users should be used and immediately discarded, on the same line, to keep that list itself confidential. The system could be devised to require a password from the user, if desired.

<u>Other Uses of a Security System</u>

A security procedure of a type described here, if it is based on a file system, is capable of providing other services as well.

First, the system can provide a monitoring of usage of the package, to any desired degree. Attempts by unauthorized users that are foiled by the security system might be recorded, partly to identify people who are trying to bust the system, and partly to identify potential customers.

Second, such a system can provide a means of communication from the owners of the package to the users. Unauthorized users can receive a polite notice of rejection, if that is desired, or notices can be posted to be read by each user the next time he validates. These notices can be anything from announcements of package modifications, to suggestions for better use of the package, to descriptions of new literature about the package.

Finally, such a system might even include a procedure for messages from users to the owners of the package, for example, requesting literature or special assistance.

A PL/1 BATCH PROCESSOR FOR APL

S. Charmonman and J. E. Bell
University of Missouri
Columbia, Missouri

ABSTRACT

This paper describes a translator for batch processing of APL. It was written in PL/1 and has been operational through the usual card reader for input and the printer for output as well as through a typewriter terminal under Remote Job Entry of the Conversational Programming System for both input and output. The subset of APL accepted by the translator is at the leevel of APL/1130. The translator provides file processing facilities via PL/1 and a form of object program for subsequent runs. It has served as a temporary substitute and then a supplement to APL/360.

Introduction

Ideally we should have interactive and batch facilities for any good high-level language. In the case of APL[9] the interactive access has been excellently provided by the interactive APL/360[10] and APL/1130[2] systems. Experimentation with algorithms and debugging of programs are best done in the interactive mode. However, once a program has been debugged and is r ly for production the source program need not and should not be reinterpreted over and over for every run. An object program or an intermediate representation should be set up for subsequent runs. If the subsequent runs are done through batch the terminal could also be used for some productive purposes instead of having its keyboard locked up to wait for the result of execution of a program.

In a non-ideal situation like at the University of Missouri in 1970 (due to reasons not in the scope of this paper) it was decided not to provide APL/360, but to provide CPS (Conversational Programming System)[1] with conversational PL/1, BASIC and remote job entry. The senior author was (and still is) strongly for APL and wanted his students to have access to APL. So, a home-grown translator for batch processing of APL was developed[4].

In order to have the translator operational as soon as possible, it was decided to use a high-level language rather than an assembly language. PL/1[7,8] was chosen for it is richer than FORTRAN IV; has been used for system programming [5,6] and was available at the University of Missouri.

The resulting translator is more than an interpreter but less than a compiler. It provides an object program not in assembly language but in a Polish form of descriptor blocks with tables of information to be used for subsequent runs if desired. Version 3.0 of the translator was compiled on the IBM 360/65 at the University of Missouri-Columbia. It runs in a batch environment with any APL program entered through a card reader and its result printed on a printer; or with both the program and the result communicated through an IBM 2741 under remote job entry mode of CPS.

Organization of the Translator

Figure 1 shows the general organization of the translator. The source program is processed through the lexical phase and syntactic phase one statement at a time to convert the original source program into a modified Polish notation. During this processing the tables of information are produced and modified.

After the source program has been completely transformed into the modified Polish notation, the execution phase executes on the modified Polish notation to produce the results of computation. Any data to the APL source program is read in during execution phase and the three tables created during the lexical phase are modified to reflect changes in the information they contain during execution of the APL source program. For simplicity it was decided that only values and not expressions would be allowed as data in version 3.0 of the translator.

During the lexical phase each atom of the APL source program is recoded into a unit of information which will be referred to as a descriptor block. A descriptor block serves as the source of all information related to the atom for which it stands.

Each descriptor block itself is logically divided into two parts. The first part, called the type section, contains 8 bits each of which may be 0 or 1. The second part, called the index section contains an integer number. Each descriptor block is exactly three bytes long, the type

section being one byte and the index section being two bytes. Figure 2 shows the logical break down of a descriptor block.

The first three bits of the type section are grouped together to form a type code. Eight type codes are possible, but only five are used presently. The bit pattern and meanings of each possible type code are also shown in Figure 2. The type code of each descriptor block is determined during the lexical phase when the descriptor block is created.

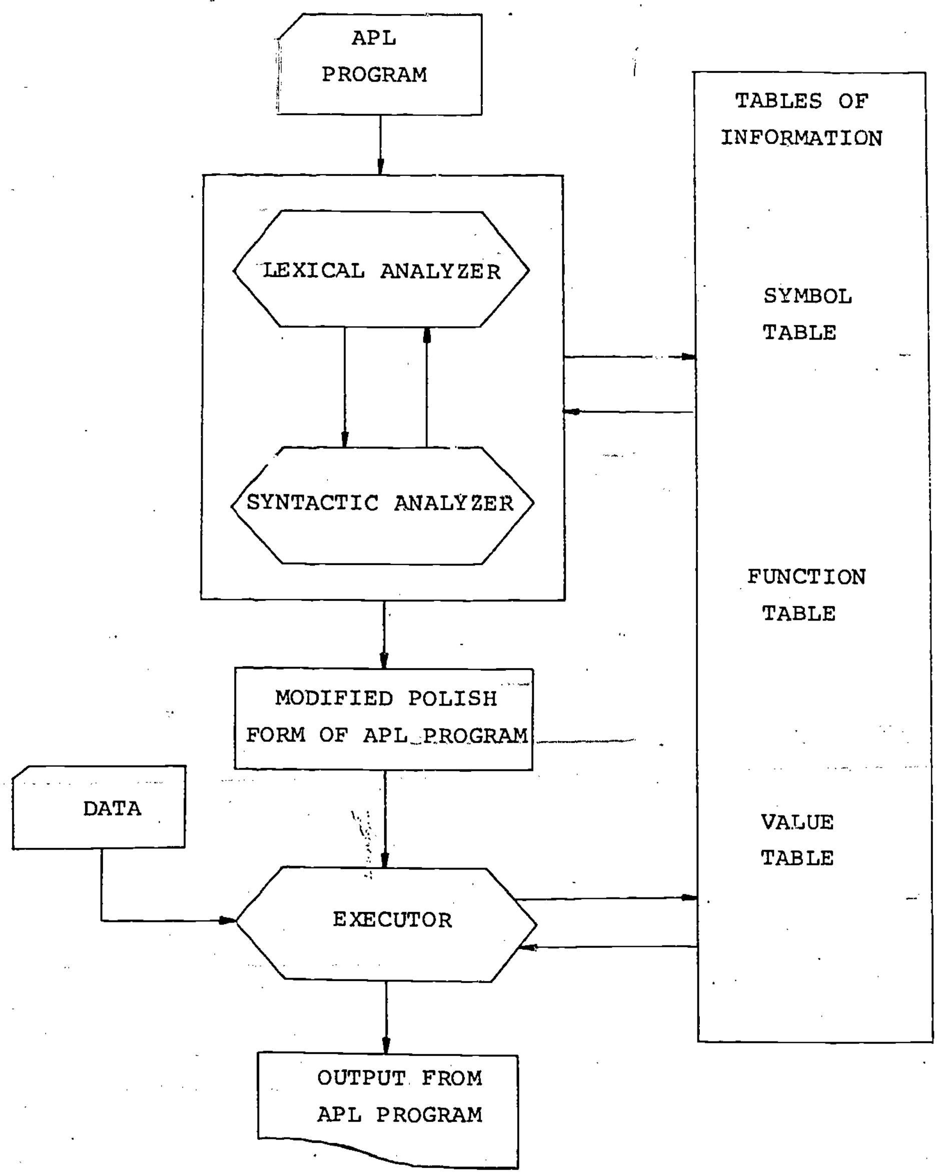
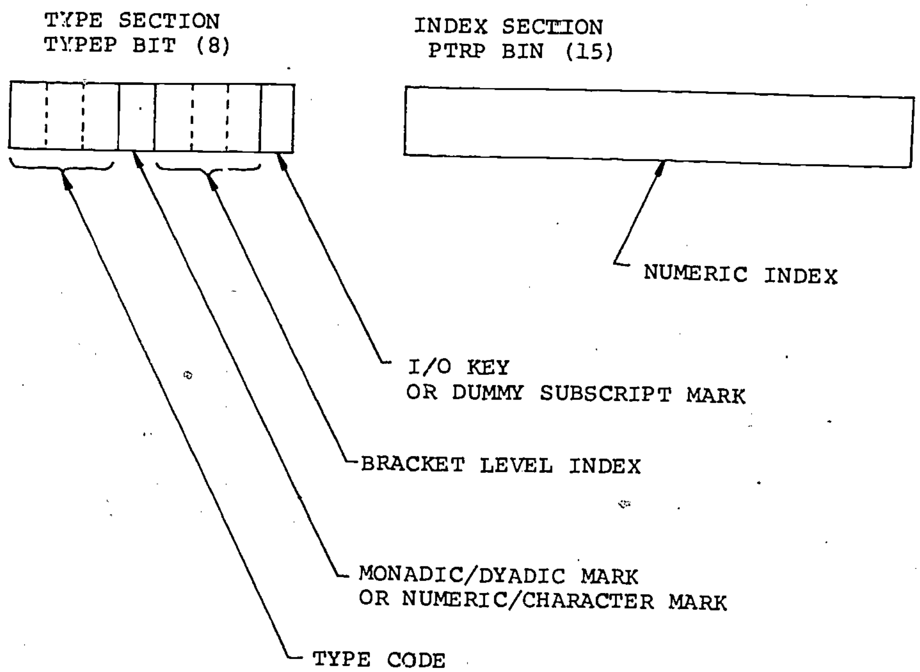


FIGURE 1: GENERAL ORGANIZATION



TYPE CODE	MEAINING
000	END OF STATEMENT
001	NOT USED
010	FUNCTION
011	CONSTANT
100	NOT USED
101	VARIABLE
110	NOT USED
111	OPERATOR

FIGURE 2: DESCRIPTOR BLOCK

Bit four may take on two different meanings depending on the type code of the descriptor block. If the descriptor block is typed as being that of a constant then bit four is marked during the lexical phase to indicate the numeric (0) or character (1) attribute of the constant. If the descriptor block is typed as being an operator then bit four is marked during the syntactic phase to indicate the monadic (0) or dyadic (1) nature of the operator.

Bits five, six, and seven are grouped together to form the bracket level index. During the syntactic phase the bracket level index is set to reflect the imbededness of each descriptor block in a subscript. The fact the the bracket level index is a three bit pattern accounts for the restriction of seven levels of subscripts in Version 3.0 of the translator.

Bit eight may take on two different meanings depending on the type code of the descriptor block. If the descriptor block is for an operator and the index section is set to indicate an input/output operation then bit eight is set during the syntactic phase to indicate whether it is an input (0) or output (1) operation. It will be noted here that the translator internally handles an I/O symbol as an operator rather than a variable. If the bracket level index is greater than zero then bit eight is set during the syntactic phase to indicate whether the descriptor block marks an actual subscript (0) or is a place marker (1) for a subscript which is implied, but does not appear, e.g., the first subscript is A[;2].

The index section of the descriptor block with its numeric index may indicate any one of several things depending on the type code found in the type section of the descriptor block. For variable-type descriptor blocks or constant-type descriptor blocks the index section contains an index to the array of pointers to the symbol table. This index may be used to index the array-of-pointer variables to the symbol table to get the pointer to a symbol-table entry and hence the symbol-table entry for the variable or constant in whose place this descriptor block stands. The index to the array-of-pointer variable to the symbol table is placed in the descriptor block during the lexical phase for constants and variables.

For function-type descriptor block the index section of the descriptor block contains an index to a pointer array to the function table. This index may be used to chain back to the function-table entry for which a function-type descriptor block stands. The index section of a function-type descriptor block is completed during the lexical phase. If the type code for a descriptor block is set to indicate an operator then the index section of the descriptor block contains the operation code as determined by an operator matrix in the lexical phase for the operator for which the descriptor block stands.

For each variable, constant, and statement label found in an APL source program during the lexical phase a symbol-table entry is created. Symbol-table entries are allocated dynamically as needed and a pointer to each allocation is kept in an array of pointer variables called PTRSE. The index section of descriptor blocks for variable, constant, and statement label contains an index to PTRSE. The limitations on the number of variables, constants, and statement labels contained within one APL source program is set by the length of the pointer array PTRSE and the area of core available for allocating symbol-table entries.

Each symbol-table entry is logically organized into five sections. The first section contains the name of the symbol represented by the symbol-table entry. The name is placed in the symbol-table entry when it is allocated and may be up to eight characters long in the present version. The second element is called the type flags and is one byte in length. Version 3.0 of the translator uses only the eighth bit to indicate whether the value associated with the table entry in question contains character (1) or numeric (0) values. The third section contains the rank and the shape of the structure. For simplicity Version 3.0 of the translator allows structures only up to the rank of three. The values -1 and -2 in the rank byte are used to indicate the empty vector and undefined structure respectively. The fourth section gives the extent of the value area and the fifth section the pointer to the value area.

Value areas are allocated and freed dynamically. If no value area has been allocated to a symbol-table entry then the extent value is set to zero and the pointer set to null.

For each function found in the APL source program, a function-table entry is created during the lexical phase. Function-table entries are created dynamically and a pointer to each allocation is placed in an array of pointers to the function-table entries (FPTR). The index section of a descriptor block for function contains an index to FPTR. The pointer array FPTR in the Version 3.0 of the translator is 100 members long and each function-table entry requires 13 bytes of core storage. Therefore, up to 100 unique functions may be used within one source program provided enough core storage is available.

Each function-table entry is logically divided into four elements. The first element contains the name of the function exactly as found in the source program. The name may be from one to eight characters long. The second element of a symbol table entry contains a type code set during the syntactic phase to indicate how many arguments will be passed into the function.

A type code of "1" indicates the presence of a right hand argument and a type code of "2" indicates the presence of both a right hand and left hand argument.

The next two elements of a function-table entry are entered into the function table during the syntactic phase and contain the line number or address of the function header and the line number of the last statement of the definition. This information is used during the execution phase for execution of the function.

The last element of a function-table entry is a pointer to a parameter list. The parameter list contains a list of indexes to the array of pointers to the symbol table (PTRSE), the parameter list is dynamically created to the length needed to contain indexes to each variable found on the function-header statement. The order of the indexes in the parameter list is significant. The first position in the list contains either an index to the symbol-table entry in which the result of the function will be found at the termination of the function, or the first position will contain zero to indicate no result will be returned by the function. The second position of the parameter will contain either the index of a local variable as listed on the function header, or the index of the right-hand argument if the function is not niladic. The third position of the parameter list contains either the index of a local variable or the index of the left-hand argument if the type code is set to two. The remaining positions of the parameter list contain the indexes to PTRSE for the remainig local variables. The minimum length of the parameter list is one for function with no argument and no local variable.

The relationship between a descriptor block and symbol table, function table and operator table is shown in Figure 3.

Sample Programs

As mentioned earlier, I/O for the translator may be either through -reader-printer or IBM 2741 terminal. With the card card-reader-printer, APL source program and output must be represented in PL/1 character set such as a modification of [3]. Through IBM 2741 the present version of the translator accepts only PL/1 character set, but a front-end is being developed to allow use of APL character set.

A sample CPS session of APL is shown in Figure 4. After the 2741 terminal has been connected to the computer a session begins by the user making a login request. The computer responds by asking for the password to be typed in by the user in the black-out spaces. If the correct password is not given by the user after a few attempts, the machine will force him out by locking the keyboard. Otherwise, it will print a message including the time and date. After this point if the machine expects you to type any line it will underscore, backspace and wait. In other words, any line you type in will appear with the first letter underscored.

In Figure 4 (a) the first command or request the user made after logging in was "load" and "list" a program segment named "aplrje". This program segment is the set of PL/1 job control cards for processing an APL program (or a batch of programs) and channelling the output to a cataloged data set to be written via the terminal.

The set of PL/1 job control cards shown in the listing of "aplrjp" is for processing APL and channelling the output to the printer rather than the typewriter terminal.

The last listing on Figure 4 (a) is a sample APL program complete with its job control cards (and not PL/1 job control cards). It has been stored under the name "a3601".

To schedule a job we use the CPS instruction

sched(A>>B>>...)

when "A, B, ..." are the names of programs which have been stored, and the symbol ">>" the PL/1 catenation operator. For example, in Figure 4 (b) after the listing of his library, the user schedules the program obtained by catenating the set of job control cards in "aplrje" and the APL segment in "a3601". The system responds by giving the job number (94 in this case) and the time it enters the queue. The status of the job may be requested by the CPS command "find(A)" where A is the job number. On the fourth line from the bottom of the listing on Figure 4 (b), the machine responds to "find(94)" that this job has been completed at 9:56:23 which is about six minutes turn-around time.

Once the job is completed the output may be printed by using the reader program which is a PL/1 program to be executed in CPS PL/1 and not CPS RJE mode. Therefore, the user must log-out from RJE as shown on the last two lines in Figure 4 (b), and log-in PL/1 as shown on the first line of Figure 4 (c). The reader program is executed by the CPS command "xeq" and it asks for the file name which is "printax" in "aplrjp".

The output of the APL program "a3601" is shown in Figure 4 (c). In general, the output from Version 3.0 of the translator is arranged in two sections. The first section is headed by a heading identifying the translator. Below the heading the APL program is reproduced and each line is numbered. Errors found during the lexical-syntactic scan are printed out below the line in which the error occurred and contain a reference to the statement in which the error occurred. Provided the program passes the lexical-syntactic scan, a message indicating that no errors were found during the syntactic scan and that the execution phase is in control is printed out.

The second section is headed by a heading which indicates the program output and the translator or system output. The output from the APL source program is printed on the left-hand side of the page. The right-hand side of the page may contain system output. System output consists of the statement number and, optionally, variable name associated with each output operation executed in the APL source program. Execution errors are printed out as they occur and will generally reference an APL source statement in which they occurred.

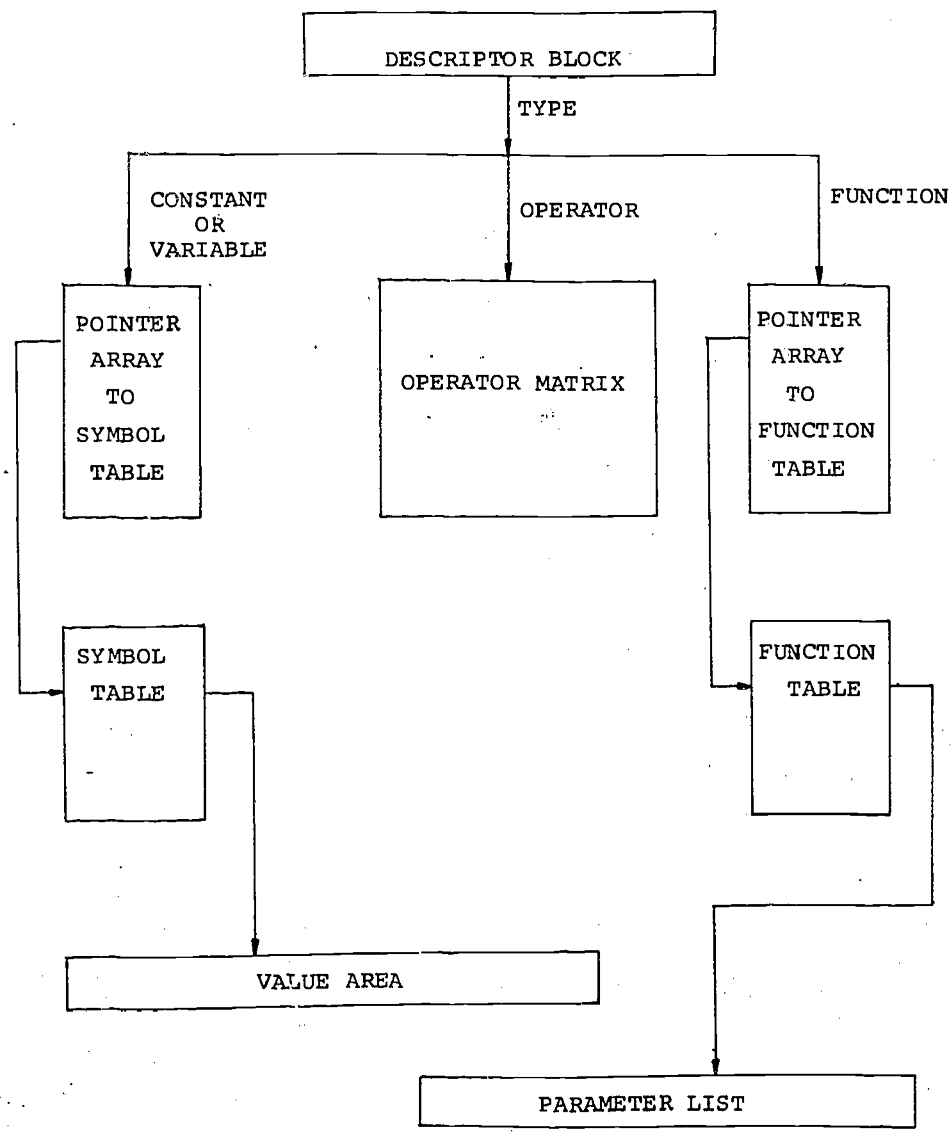


FIGURE 3: RELATIONSHIP BETWEEN A DESCRIPTOR BLOCK AND TABLES

```
login(cps003,n24,rje)
PASSWORD: [redacted]
GOOD MORNING; USER 01; TIME   9:40:54  5/27/71;
load(aplrje)
list
 10.        0100 //XJOBCARD JOB (K11197,,280K),CHARMONMAN,MSGLEVEL=(1,1),REGION=310K
 20.        0200 //S1 EXEC AMPLE
 30.        0300 //SYSPRINT DD DSN=CPS003,N24,PRINTAX,DISP=(,CATLG),
 40.        0400 //    UNIT=2314,SPACE=(TRK,(1,1)),VOL=SER=MFC163
 50.        0500 //APL.SYSIN DD *
load(aplrjp)
list
 10.        0100 //XJOBCARD JOB (K11197,,280K),CHARMONMAN,MSGLEVEL=(1,1),REGION=310K
 20.        0200 //S1 EXEC AMPLE,TIME=2
 30.        0300 //SYSPRINT DD SYSOUT=A
 40.        0400 //APL.SYSIN DD *

load(a3601)
list
 10.        0100 )JOB
 20.        0200           3 @* 4                @.
 30.        0300           X :< 3 @* 4           @.
 40.        0400           X                     @.
 50.        0500           Y :< _5               @.
 60.        0600           X@+Y                  @.
 70.        0700           P :< 1 2 3 4          @.
 80.        0800           P @* P                @.
 90.        0900           P @* Y                @.
100.        1000           Q :< 'CATS'           @.
110.        1100           Q                     @.

120.        1200 )DATA
130.        1300 )END
140.        1400 /*
```

(a) JOB CONTROL CARDS AND A SAMPLE PROGRAM

```
lib list
*brm    *xbrm   *desc   *al     watjcl  wattst  watdat  pascal
 a3601   cardd   facmn   apljcl  apll    jclwat  jclapl  aplxxx
 aplpg   apljob  aplrje  aplx4   a3602   nodata  pscld   aplmod
*aplrd   aplpg1  aplpg2  aplrjp  aplpg3  aplpg4  facfun  permt
 job     endjob  fnmain  expand

sched(aplrje||a3601)
JOB CPSJOB94 ENTERED QUEUE 00 AT  9:50:18 71.147
logout
TIME  9:50:54; TIME USED: CPU 00:00:06; TERM 00:09:59; PAGE 00:09:50;

login(cps003,n24,rje)
PASSWORD: [redacted]
GOOD MORNING; USER 01; TIME   9:55:52  5/27/71
find(94)
JOB CPSJOB94 COMPLETED AT  9:56:23 71.147
logout(resume)
TIME  9:56:41; TIME USED: CPU 00:00:02; TERM 00:00:49; PAGE 00:00:41;
```

(b) SCHEDULE AND FIND

FIGURE 4: A SAMPLE CPS SESSION (CONTINUED)

```
login(cps003,n24)
PASSWORD: [redacted]
GOOD MORNING; USER 01; TIME   9:57:03  5/27/71;
load(reader)a59sys
xeq
Enter simple file
name
'printax'
VERSION 3.0                         APL/UMC
                          COMPUTER SCIENCE DEPARTMENT
                        UNIVERSITY OF MISSOURI, COLUMBIA
      1    3 @* 4                                                @.        0200
      2    X :< 3 @* 4                                           @.        0300
      3    X                                                     @.        0400
      4    Y :< _5                                               @.        0500
      5    X@+Y                                                  @.        0600
      6    P :< 1 2 3 4                                          @.        0700
      7    P @* P                                                @.        0800
      8    P @* Y                                                @.        0900
      9    Q :< 'CATS'                                           @.        1000
     10    Q                                                     @.        1100
COMPILATION COMPLETE
*****NO ERRORS ENCOUNTERED IN SYNTACTIC SCAN, NORMAL PROCESSING CONTINUING
*****EXECUTION PHASE NOW IN CONTROL*****
*****************************************************<--PROGRAM OUTPUT
                                                     APL/UMC SYSTEM OUTPUT
                                                                V

                                                     STATEMENT#  | VAR NAME
                               12                    ---->   1
                               12                    ---->   3     X
                                7                    ---->   5
                                1   4   9  16        ---->   7
                               _5 _10 _15 _20        ---->   8
                             CATS                    ---->  10     Q
                                            ***SAMPLE PROCESSING COMPLETE***
```

(c) OUTPUT ON THE TERMINAL

FIGURE 4: A SAMPLE CPS SESSION (CONTINUED)

CPS editing facilities may be used to edit any APL program. A sample editing session is shown in Figure 5 with comments on the right-hand side of the CPS listing.

One of the assets of CPS is the ability to store user-defined functions in the CPS data set. On scheduling an execution, any set of these functions can then be concatenated onto any source program requiring them.

Other sample executions of the translator are presented in Figure 6 and 7.

Concluding Remarks

The batch processing translator presented has been operational on the IBM 360/65 at the University of Missouri, first as a temporary substitute for a year and then as a supplement to APL/360. As a substitute it provides access to a form of APL for teaching and research. It allowed a local group of APL users to be set up and became a factor in the university's decision to make APL/360 available in 1972. As a supplement it allows access to APL when terminals are occupied for ATS and CPS. It also allows programs to be entered through the card reader, edited on the terminal, sample output checked on the terminal, and final output printed on the printer if desired.

Although the subset of APL accepted by Version 3.0 of the translator is at the level of APL/1130 the translator does provide file processing facilities via PL/1 and CPS RJE. It also provides object code in the form of Polish strings of descriptor blocks and information tables for possible uses in subsequent runs.

One of the drawbacks of the translator is, of course, the character set. This problem would be solved if and when an an APL print chain becomes available. With a minor modification of the translator, a program may be entered on IBM 2741 with APL type ball and program listing done on the terminal in APL. If the numerical result of computation is voluminous, it may be printed on the printer, leaving the terminal available for other uses.

```
login(cps003,n24,rje)
PASSWORD: [redacted]
GOOD MORNING; USEP 02; TIME  11:12:15  5/28/71;
10.        N :< #=                                  @.    ERASE ANY CHARACTER BY
5.         #= :< 'PASCAL STARNCLE'                  @.    BACKSPACING OVER IT.
5!
  5.       #= :< 'PASCAL START'                     @.    TO SEE A LINE, TYPE ONE
list                                                      OF THE FOLLOWING TWO:
  5.          0050  #= :< 'PASCAL START'               @.    <LINE NO>!
 10.          0100  N :< #=                            @.    <ATT>LIST
#20,10     #= :< P :< I :< 1                        @.
 30.       A:= #= :< P :< (0 @, P) @+ P @, 0        @.    AUTOMATIC LINE NUMBERING.
 40.       :> (N @> I :< I @- 1) #/ B               @.
 50.       #= :< 'END OF SAMPLE PROGRAM'            @.
 60.
list 5 thru 10                                            LIST SPECIFIC SECTION OF
  5.          0050  #= :< 'PASCAL START'               @.  PROGRAM.
 10.          0100  N :< #=                            @.
40!*M*N*                                                  REPLACE A SUBSTRING IN A LINE.
 40.      :> (N @> I :< I @- 1) #/ B                @.
40!/@-/@+/B/A/                                            REPLACE SEVERAL SUBSTRINGS.
 40.      :> (N @> I :< I @+ 1) #/ A                @.
45.       #= :< 'PASCAL END'                        @.
list 45 thru 50
 45.          0450  #= :< 'PASCAL END'                 @.
 50           0500  #= :< 'END OF SAMPLE PROGRAM'      @.
erase 50 thru 50                                          ERASE A NUMBER OF LINES.

list
  5.          0050  #= :< 'PASCAL START'               @.
 10.          0100  N :< #=                            @.
 20.          0200  #= :< P :< I :< 1                  @.
 30.          0300  A:= #= :< P :< (0 @, P) @+ P @, 0  @.
 40.          0400  :> (N @> I :< I @+ 1) #/ A         @.
 45.          0450  #= :< 'PASCAL END'                 @.

 save(pascal)
```

FIGURE 5: A SAMPLE EDITING SESSION

```
VERSION 3.0                                           APL/UMC
                                             COMPUTER SCIENCE DEPARTMENT
                                          UNIVERSITY OF MISSOURI, COLUMBIA
        1     $+ R:< "PASCAL" N                          @.                0020
        2     #= :< P :< I :< 1                          @.                0030
        3     A:= #= :< R :< P :<(0 @, P) @+ P @, 0      @.                0040
        4     :> (N @> I :< I @+ 1) #/ A                 @.                0050
        5     $+                                         @.                0060
        6     R :< "PASCAL" 5                            @.                0070
COMPILATION COMPLETE
*****NO ERRORS ENCOUNTERED IN SYNTACTIC SCAN, NORMAL PROCESSING CONTINUING
*****EXECUTION PHASE NOW IN CONTROL*****
*****************************************************************<--PROGRAM OUTPUT
                                                                 APL/UMC SYSTEM OUTPUT

                                                                 STATEMENT# | VAR NAME
     1                                                           ---->    2
     1      1                                                    ---->    3
     1      2      1                                             ---->    3
     1      3      3      1                                      ---->    3
     1      4      6      4      1                               ---->    3
                                                        ***AMPLE PROCESSING COMPLETE***
```

FIGURE 6: A USER'S DEFINED FUNCTION

```
VERSION 3.0                                              APL/UMC
                                             COMPUTER SCIENCE DEPARTMENT
                                          UNIVERSITY OF MISSOURI, COLUMBIA

IN :< 'X*(Y+Z+(K/L)/B)+E*B/(A/(J*K*L)+Z)@'       @.
K:< 1 @.
 ST :< $| $? IN @.
  ST(%K%) :<'@' @.
   SP :< 0 1 2 3 0_1000 @.
    IP :< 4 1 2 3 1_1000 @.
     OP :< '(+*/)@' @.
      OD :< 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' @.
       OT :< $| I :< 0 @.
 L0:= I :< I @+ 1 @.
    :> (¬+ IN(%I%) $& OD) #/ L1 @.
    OT :< OT @, IN(%I%) @.
#= :< OT @.
    :> L0 @.
L1:= :>(IN(%I%) @= ')' ) #/ L2 @.
    :> (IN(%I%) @= '@' ) #/ L3 @.
L7:= XX :< OP $| ST(%K%) @.
   YY :< OP $| IN(%I%) @.
   :> (SP(%XX%) ¬< IP(%YY%) ) #/ L4 @.
   K :< K @+ 1 @.
   ST(%K%) :< IN(%I%) @.
#= :< ST @.
   :> L0 @.
L4:= OT :< OT @, ST(%K%) @.
    K :< K @- 1 @.
    :> L7 @.
L2:= :>(ST(%K%) @= '(' ) #/ L5 @.
    OT :< OT @, ST(%K%) @.
#= :< OT @.
    K :< K @- 1 @.
    :> L2 @.
L5:= K :< K @- 1 @.
    :> L0 @.
L3:= :>(¬+ ST(%K%) @= '@' ) #/ L6 @.
    :> L8 @.
L6:= OT :< OT @, ST(%K%) @.
    K :< K @- 1 @.
    :> L3 @.
L8:=  #= :< 'THE FINAL POLISH STRING IS :' @.
    #= :< OT @.

COMPILATION COMPLETE


*****NO ERRORS ENCOUNTERED IN SYNTACTIC SCAN, NORMAL PROCESSING CONTINUING


*****EXECUTION PHASE NOW IN CONTROL*****
```

FIGURE 7: A SHUNTING ALGORITHM

```
******************************  **********<--PROGRAM OUTPUT

                                        APL/UMC SYSTEM OUTPUT
                                                  V

                                        STATEMENT# | VAR_NAME

X                                       ---->  13   OT
@*                                      ---->  22   ST
@*(                                     ---->  22   ST
XY                                      ---->  13   OT
@*(+                                    ---->  22   ST
XYZ                                     ---->  13   OT
@*(+                                    ---->  22   ST
@*(+(                                   ---->  22   ST
XYZ+K                                   ---->  13   OT
@*(+(/                                  ---->  22   ST
XYZ+KL                                  ---->  13   OT
XYZ+KL/                                 ---->  29   OT
@*(+//                                  ---->  22   ST
XYZ+KL/B                                ---->  13   OT
XYZ+KL/B/                               ---->  29   OT
XYZ+KL/B/+                              ---->  29   OT
@+(+//                                  ---->  22   ST
XYZ+KL/B/+*E                            ---->  13   OT
@+*+//                                  ---->  22   ST
XYZ+KL/B/+*EB                           ---->  13   OT
@+*///                                  ---->  22   ST
@+*/(/                                  ---->  22   ST
XYZ+KL/B/+*EBA                          ---->  13   OT
@+*/(/                                  ---->  22   ST
@+*/(/(                                 ---->  22   ST
XYZ+KL/B/+*EBAJ                         ---->  13   OT
@+*/(/(*                                ---->  22   ST
XYZ+KL/B/+*EBAJK                        ---->  13   OT
@+*/(/(*                                ---->  22   ST
XYZ+KL/B/+*EBAJK*L                      ---->  13   OT
XYZ+KL/B/+*EBAJK*L*                     ---->  29   OT
@+*/(+(*                                ---->  22   ST
XYZ+KL/B/+*EBAJK*L*/Z                   ---->  13   OT
XYZ+KL/B/+*EBAJK*L*/Z+                  ---->  29   OT
THE FINAL POLISH STRING IS :            ---->  39
XYZ+KL/B/+*EBAJK*L*/Z+/*+               ---->  40   OT

                    ***AMPLE PROCESSING COMPLETE***
```

FIGURE 7: A SHUNTING ALGORITHM (CONTINUED)

REFERENCES

1. Andrada, J. E., et al, Conversational Programming System, Program No. 360D-03.4.016, IBM Hawthorne, New York, 1969.

2. Berry, P. C., APL/1130 Primer, Form No. GC20-1697, IBM Corporation, 1968.

3. Charmonman, S., "Sixty-Character Representation of APL Symbols," APL Quote Quad, Vol. 2, No. 2 (July 1970), pp. 5-10.

4. Charmonman, S., Bell, J. E., Browns, W. J., McGee, P. A. and Simmons, C. R., APL/UMC: An Experimental Translator for Batch Processing of a Subset of APL, Technical Report No. 2 Department of Computer Science, University of Missouri-Columbia, May, 1971.

5. Graham, R. M., Use of High Level Language for System Programming, Report No. MAC TM-13 MIT, Cambridge, Massachusetts, September 1970.

6. Hedrick, G. E., An Implementation of a PL/1 Systems Program which Demonstrates the Feasibility of Systems Programming in a High Level Language, Report No. IS-2244, Iowa State University, Ames, Iowa, January 1970.

7. IBM System/360 Operating System PL/1 (F) Programmer's Guide, GC28-6594-6, June 1970 .

8. IBM, IBM System/360 Operating System PL/1 (F) Language Reference Manual, GC28-8201-3, July 29, 1970.

9. Iverson, K. E., A Programming Language, John Wiley and Sons, Inc., New York, 1962.

10. Pakin, Sandra, APL/360 Reference Manual, Science Research Associates, Chicago, 1968.

SUBTASKING IN APL

Alain Miville-deChêne and Louis P. A. Robichaud
Université Laval
Quebec, P. Q., Canada

Introduction

In this paper we discuss a modificaiton to APL/360 which allows rather interesting modes of use of APL, such as subtasking, multitasking, working without a terminal, communicating between terminals synchronously or asynchronously, etc.

By subtasking we mean the subdivision of a main program into parts called subtasks, which may be executed concurrently, permitting such things as the overlapping of input/output with processing. One might consider that multitasking is involved in a situation where a number of APL users are controlled by the same APL task.

In APL each signed-on user executes only one program at a time, although APL is a system in which a number of users are (conceptually) working concurrently. However they are essentially working independently of one another, except when sending messages.

APL contains the basic elements for our subtasking needs. However subtasking requires a more sophisticated means of control and communication between tasks, as well as the ability of starting and stopping tasks.

Basic Concepts

If one user could be connected to another user's workspace, he would then have complete control of what is done in that workspace.

The functions:

```
SL ← SEND 'TXT'

Z ← RD SL
```

allow communication to port SL just as if our terminal was physically connected to port SL. It is important to note that the text 'TXT' is seen by port SL exactly as if it were being typed on its own terminal.

The output produced by port SL in response to 'TXT' can be read and assigned to a character variable Z. RD will read only one line of output at a time. It is used in the function:

```
Z ← READ SL
```

which will read all the output from port SL and remove characters such as the 6 blanks produced by APL. For example:

```
      60 SEND')LOAD 1234 WS
⎕←R←ι5
)SAVE'
      READ 60
SAVED  14.05.02 05/11/72
1  2  3  4  5
 15.02.05 05/11/72 1234 WS
      )COPY 1234 WS R
SAVED  15.02.05 05/11/72
      R
1  2  3  4  5
```

In order to make the system more practical and to avoid interfering with a real user it was necessary to be able to automatically sign-on other ports, hereafter called SLAVES. The port which causes the sign-on is called the MASTER port, and it can have a number of SLAVES. Each slave has the same user-identification and workspace quota, as the master, and is seen by the APL system as a normal user. Example:

```
      )PORT AMD
013 AMD
      GS
62
      )PORT AMD
013 AMD
062 AMD
      62 SEND')OFF'
```

where GS (get slave) is a function causing the sign-on of a slave port (by a master or by a slave). This operation is extremely fast (a few milliseconds) since there is no need for the system to validate an account number.

It must be carefully noted that master and slaves form a group, completely independent from any other master-slave group in the system. The master always has complete control over his slaves and he can not affect other user's slaves except when specifically authorized by these users - a process called SHARING, to be discussed later.

Buffer Supervision

Each port has a block of information called PERTERM describing the state of that port and containing information pertaining to the signed-on user (account number, initials, etc...)

APL/360 uses a dynamic buffer allocation scheme. All I/O buffers are grouped in a buffer pool and are linked together to form chains.

Each PERTERM has pointers to two chains of buffers, one each for input and output. The interpreter gets the character string to be analyzed from the input chain. When an input line has been analyzed, the buffers used to hold that line are returned to the buffer pool.

All of the slaves' I/O goes through the buffers, each of which has space for 20 bytes of outgoing data and 19 bytes of incoming data. The system allocates a maximum of 20 buffers to any port. With such limitations, a certain amount of control over buffer allocation becomes necessary, since a slave attempting to use more than 20 buffers will be suspended until enough buffers are available to continue execution.

The following functions are used to control buffers:

R ← *BFA SL*	–	returns the number of buffers currently allocated to slave *SL*.
NOOUT SL	–	causes the system to ignore output requests from port *SL* . Text already in output buffers is not affected and can be read at any time.
OUT SL	–	reverses the effect of *NOOUT*
FO SI	–	frees all buffers in the output chain of port *SL*.

Proper use of the above functions can avoid all problems of buffer allocation.

Synchronisation and Interrupts

Each PERTERM has a full word called the "global" variable whose value is independent of the users' workspace.

R ← ⊛1	–	returns the value of this variable without changing it.
R ← ⊛1,*N*	–	changes its value for N and returns the old value.

The symbol ⊛ is the primitive operator affectionately called "GLOBUL" (1,2) which is used in monadic and dyadic form for a number of special purpose functions.

SL SYNC ARG is used to synchronize tasks. It tests bits or values in the global variable of another port and, depending on the result of the test, either falls through to continue execution or enters the wait state. Testing is retried every 1/2 second until the conditions are met. (Fig. 1)

ARG is a vector of 4 or 5 elements used to build two machine language instructions, one test, and one branch.

```
                    MASTER                        SLAVE(62)
                      .                              .
                      .                            ⊗1 0
                      .                              .
.............         .                              .
|CLI GLBL+3,1|→     62 SYNC 2 3 1 8                  .
|BCR 8,RETRN |→       .                              .
.............↑        .                              .
             ↑        .                              .
SUSPENSION→→→→↑       .                              .
                      .                              .
RESUMPTION →→→→→      .                            ⊗1 1
                      .                              .
```

FIG 1: A SIMPLE EXAMPLE OF SYNCHRONISATION

```
      ⍝ TERMINAL 1

      )LOAD TEST
SAVED  17.12.18 05/11/72
      GS
62
      62 SHARE 62
      12 SHARE 62
      62 SEND')LOAD TEST
NOOUT 62
TO 62
COPLIB'
      62 1 TEST'7000'
17.14.21
17.14.22
062 AMD: TRANSFERING  17.14.33
17.15.00
      )LIB 987
WS1
WS2
WS3
      )LOAD 987 WS1
SAVED  17.14.35 05/11/72
      )LOAD 987 WS2
SAVED  17.14.46 05/11/72
      )LOAD 987 WS3
SAVED  17.14.58 05/11/72
      )PORT AMD
012 AMD
029 AMD
      ⍝ THE SLAVE ON PORT 62 COMMITTED SUICIDE.
```

```
⍝ TERMINAL 2
      )LOAD 314158 TEST
SAVED  17.12.18 05/11/72
      62 2 TEST'987'
17.13.45
17.14.34
17.14.59
```

FIG 2:OUTPUT ON THE TWO TERMINALS CONNECTED TO THE SLAVE WHICH EXECUTES COPLIB.

ARG[1]	type of comparison: 0 CLC compare logical character 1 TM test under mask 2 CLI compare logical immediate
ARG[2]	position (0,1,2 or 3) from left of the first (or only) byte of the comparison.
ARG[3]	immediate mask for CLI and TM or length of the comparison for CLC. (The length must be ≤4 - *ARG*[2])
ARG[4]	condition code used in a BCR instruction. If the branch is taken, testing is stopped and execution of the program continues.
ARG[5]	constant to compare with in the CLC instruction (*ARG*[1]↔0)
ATTN SL	causes an attention on port *SL*.
SPIE LN	intercepts errors occuring during the execution of programs. When an error occurs (SYNTAX, RANK, ETC...), the system automatically branches to line LN in the program.
R←FSW	returns an integer vector of length two: error code, line in which the error occured.
INT SL	generates an 'INTERRUPT' error when slave SL is executing a program. This error occurs just before starting the execution of a new line in the slave's program. If the SPIE function was executed, the slave can process the interreupt and the execution of:

$$\rightarrow {}^{-}1 \uparrow FSW$$

will return to the point of interruption. The interrupt is prevented from occuring in the middle of a line in order that statements of the type:

$$A \leftarrow (I\leftarrow I+1)\ \rho A$$

do not get executed twice.

SHARING SLAVES is accomplished by:

WHO SHARE SL

The master uses this fuction to permit port WHO to use his slave SL. For the time being, a SHARER can use all of the slave functions on a shared slave. A function will soon be implemented in order to limit a sharer's access to only one or more of the slave functions as decided by the master.

There is an interesting case when a slave is shared on itself. It can then send itself input and read its own output - a gross simulation of the ∊ (unquote) function.

WORKING WITHOUT A MASTER:

When the master ends his session, all his slaves are normally forced off the system. There are some applications where a program is essentially CPU bound and monopolizes the use of a terminal for nothing.

KEEP SL permits keeping a slave signed-on and working even when the master has signed off. In a subsequent sign-on the master can see if the slave's program is progressing normally.

UNKEEP SL reverses the effect of KEEP.

A more powerful use of this stand-alone mode of operation can be had by sharing the slave on itself. The slave can then send itself involved sequences of commands, analyze its own output and possibly correct some errors.

T EXPRESS SL forces a sign-off of port SL in T minutes. This function is normally applied to stand-alone slaves, in order to make certain that the slave does not get caught in the system.

UNEXPRESS SL reverses the effect of EXPRESS.

A simple problem is given in the appendix in order to illustrate a number of the functions described above.

Conclusion

The system presented here is in an early stage of development. However, even at this stage, it extends the usefulness of APL in our environment. In the near future we plan to add such things as: slave quotas-analogous to workspace quotas, and limitations on the access of sharers. It will be possible to reserve slaves for certain time periods to make certain that slaves are available.

REFERENCES

Colloque APL, Paris 9-10 septembre 1971
Institut de Recherche d'informatique et d'automatique
Domaine de Voluceau - 78 Rocquencourt - France.

1. G. Dhatt, L. P. A. Robichaud. APL, flow graphs and finite elements. p. 37-69.

2. P. H. Fortin, D. Samson, P. Laverdiere, L. P. A. Robichaud. Utilisation d'APL dans le cadre du projet des statuts du Quebec. pp. 115-137.

APPENDIX

This example is not one of a typical application, but rather a concise presentation of many of the slave functions. It illustrates the following points. (see figures 2 and 3).

1. Getting a slave
   a. the master gets a slave
   b. a slave gets a slave

2. Sharing a slave
   a. with another port
   b. on itself

3. Sending input to a slave for execution
   a. master to slave
   b. sharer to slave
   c. slave to slave

4. Synchronisation
   a. master-slave
   b. sharer-slave
   c. slave-slave

5. Reading the output of a slave

The COPLIB function, executed by a slave, copies all the workspaces of a library into another (or the same). There are two users communicating with the slave: the master who sends the identificaiton of the source library, and a sharer who provides the identification of the sink. Synchronisation is necessary to make sure that the value of the sink is not sent before that of the source.

The left argument of the TEST function is a two elements numeric vector consisting of the port number of the slave and a code: 1 for the master, 2 for the sharer.

The right argument is a library number - the source or the sink depending on who is executing the function. Three time-values are printed:

1. entry into the function
2. transmission of input for the slave
3. the end of the library copy operation

The EXEC function executes a series of system commands. The slave first sends itself (port number: ⊗6) the system commands as input and a 0. It then executes a ⎕ . The first system command is read and executed. The ⎕ again asks for input. This continues until a 0 is read. This 0 becomes the result of the ⎕ and the EXEC is terminated. For example:

```
      R←⎕
⎕:
      )SAVE TEST
 12.57.09 05/10/72
⎕:
      )LOAD WS
SAVED  23.12.40 05/09/72
      ⍝ HERE WE CAN DO ANYTHING
      )LOAD TEST
SAVED  12.57.09 05/10/72
⎕:
      0
```

```
     ∇ COPLIB;R;S;SOU;SINK;A;L
[1]    ⍝ PERMIT THE FIRST TERMINAL TO SEND INPUT FOR THE UPCOMMING ⎕
[2]    ⊗ 1 1
[3]    ⍝ GET A SLAVE AND SEND IT )LIB 00000 .
[4]    ⍝ THE ⊗1 1 IS TO AVOID RACE CONDITIONS... WE COULD TRY TO READ
[5]    ⍝ THE SLAVE'S OUTPUT BEFORE IT HAD STARTED TO EXECUTE THE )LIB
[6]    (S←GS) SEND ')LIB ',(SOU←DEC ¯1↑P←⎕),CR,'⊗1 1'
[7]    ⍝ L IS GOING TO BE A N×11 MATRIX, THE RESULT OF )LIB
[8]    L← 0 11 ⍴' '
[9]    ⍝ WAIT FOR THE ⊗1. (CLI GLOBVAR+3,1 :BCR 8,RETURN)∇
[10]   S SYNC 2 3 1 8
[11]   →(' '∧.=6↑R←RD S)/2+⍳26
[12]   →(¯1+⍳26),⍴L←L,[⍳1] 11↑R
[13]   ⍝ WE DON'T NEED THE SLAVE ANY MORE.
[14]   S SEND ')OFF'
[15]   EXEC ')MSGN ',(DEC(⊗ 1 2)⍴P),' TRANSFERING  ',TIME
[16]   SINK←DEC ¯1↑P←⎕
[17]   ⍝ TRANSFER THE WORKSPACES
[18]   EXEC ')SAVE ',CR,')LOAD ',SOU,' ',A,CR,')WSID ',SINK,(A←' ',, 1 11
       ↑L),CR,')SAVE',CR,')LOAD TEST'
[19]   →(×1⍴⍴L← 1 0 ↓L)/¯1+⍳26
[20]   ⍝ COMMIT SUICIDE
[21]   EXEC ')OFF'
     ∇
     ∇ EXEC BLA
[1]    (⊗6) SEND BLA,CR,'0'
[2]    ⍝ INPUT WILL BE EXECUTED IN THE ⎕
[3]    BLA←⎕
     ∇
     ∇ N TEST LIB
[1]    TIME
[2]    (N←1⍴N) SYNC 1 3 ,(1↓N),1
[3]    N SEND(DEC⊗6),' ',LIB
[4]    TIME
[5]    N SYNC 2 3 0 8
[6]    TIME
     ∇
     ∇ R←DEC N
[1]    R←'0123456789'[(⍳1)+(10⍴10)⊤''⍴N]
     ∇
```

FIG 3:LISTING OF PERTINENT FUNCTIONS.

SUGGESTION FOR A "MAPPED" EXTENSION OF APL

C. Leibovitz
University of Alberta
Computing Center

Users of APL are under the "spell" of beauty, conciseness and elegance of the language. They have however to go back to Fortran, for instance, whenever they cannot avoid a loop executed a great number of times in order to limit the CPU time used.

The natural desire for enlarging the class[1] of cases in which "it would pay" to use APL, is the origin of a great number of suggestions for modifications to and extensions of APL.

However, an APL interpreter is a complicated collection of interrelated software forming a unity that should not be disrupted. A modification in any part of the collection will have repercussions on the operation of the remainder of the programs and there is no a priori reason preventing these repercussions from being harmful and in need of necessary corrections that may not be welcomed (if at all possible).

It is therefore not enough to show that a given modification is needed; it is necessary to show that it is indeed implementable and has no disruptive character.

Our proposed modification is, in a sense, a "mapping" of an actual interpreter. The logical structure of the mapping is such that we may conclude that: "if there exists an APL interpreter that works, then our mapping will work too."

Review Of A Non-Modified APL Interpreter

Each time a line is entered from a terminal, the interpreter checks the nature of the line: is it for instance a command? or a line in definition mode? or in execution mode?... Let us designate by CHECK the module of the interpreter that finds out the nature of a line and decides what other module is to handle the line. If the line has been entered in the execution mode, it will be executed from right to left. However, for a number of reasons, this cannot be a straightforward procedure:

1. There is no one-to-one correspondence between a "primitive mathematical symbol" and an execution routine. One same symbol may be a monadic function or may be a dyadic one.

2. The mathematical meaning of a symbol may depend on the nature of another symbol placed at its left.

3. There may be brackets altering the normal right-to-left order of operations.

4. There may be mistakes in the line making it unexecutable.

There must therefore exist a module that will analyse the line, will call execution routines in a proper order and provide those routines with the values of the variables.

We are not concerned here with the way in which this is done, it is enough for us to know that it is actually done, i.e. there exists in the interpreter a module, we call it GRAM, that takes care of a line in execution mode. GRAM issues "orders" for space, for fetching values for parameters and variables, for erasing intermediate unnecessary results, for storing needed intermediate results, for finding out which routine is to be called, for calling it, for issuing error messages.

We thus designate by GRAM all the parts of the interpreter that stand between a line recognized in execution mode, and its actual execution. Everything the computer does in execution mode is therefore the consequence of "orders" issued to the computer by GRAM while analysing an entered line in execution mode.

The Need For A Modification

The correct execution of a statement results from the collection of correct "orders" issued by GRAM in a correct sequence. However, the main work done by GRAM is not so much to issue those orders but to find out which orders are to be issued.

In the MAPPED LEVEL (the name we give to our APL modified version), the function is to be stored in such a way that the orders to be executed, and their proper sequence, is known in

advance. The execution of the function thus becomes faster because there is no need for syntax checking time.

Mapped Level

We recall that CHECK examines the nature of a line and delivers it to GRAM if the line is in execution mode. CHECK has of course other alternatives than calling GRAM. We will not modify the existing alternatives; we will add one alternative more that we call Mapped Level. It means that once a line is entered, CHECK will ask an additional question: is it a mapping command? A negative answer will result in the unmodified procedure going on. A positive answer will result in a modified procedure described below.

It may be possible later to allow the use of the mapping command to all users. However, in order to simplify our discussion we will consider the case in which the mapping command is available to a priviledged APL user.

Using the mapping mode, the user can form a library of "mapped functions" that cannot be edited or modified but can be executed by any APL user.

When the user issues the mapping command, he must add two "parameters" which are the name of the unmapped function and the name under which the mapped function will be stored. The function to be mapped either does not call for another function or calls for a number of functions that have already been mapped. The list of all mapped functions is stored in the symbol table in the workspace of the priviledged user.

The mapping command will "deliver" the function to be mapped to a module we call MAPGRAM to indicate that, in a sense, this module is a mapping of the GRAM module.

MAPGRAM will proceed to analyse the lines of the function in the way GRAM would have done it with the following differences.

1. MAPGRAM considers all symbolic names as defined and does not issue value-error messages. Every symbolic name is compared with the symbol table of mapped functions. Depending if the symbolic name exists or does not exist in the table, MAPSYNT will respectively consider it a defined function or variable.

2. MAPGRAM will analyse lines of a function already tested in the unmapped mode by the user. This function is supposedly syntax-error free (this concept will be discussed later.) Therefore, for proper values of the arguments, GRAM would have issued a number of "orders": fetch, store, reserve storage place, call for execution routine, erase, etc...

MAPGRAM will issue "mapped" orders that could be described by: "copy and store in proper order the 'orders' that GRAM would have issued." For instance, whenever GRAM would have called for storage, MAPGRAM will order to store a copy of the call for storage space; wherever GRAM would have called for a given execution routine, MAPGRAM will order to store a copy of this execution routine.

In short, the mapping of the function will consist of the collection in proper order of copies of fetching routines, store routines, execution routines, etc...

These routines will be linked either by MAPSYNT or by the module LINK active at execution mode for mapped functions. The linkage consists of taking care of the proper order and of the addresses of the intermediate results and transforming the copy of a call into an actual call of a routine. It must for instance insure that the output address of a given execution routine may have to be identical with the input address of the next execution routine.

In short LINK takes care of a mapped function in the execution mode. LINK is called every time the name of a mapped function appears in a line at execution time.

Error Messages

APL delivers two kinds of error messages. The first kind corresponds to what we call a "built-in error". It is delivered when GRAM concludes that there does not exist an execution routine corresponding to the symbols entered in the line. This kind of error will be delivered for instance if there is, at execution time, a symbolic name not yet defined or if a line is entered with mathematical symbols in a non-sensical sequence. The second kind of error messages is delivered by an execution routine when GRAM does find out, at a given stage of execution, that execution routine is to be called and when this routine cannot be executed for the values and number of arguments delivered to it (rank error and domain error for instance).

The built-in errors can be detected during the mapping operation by MAPGRAM in exactly the same way as GRAM is doing it, i.e. by taking over in MAPGRAM the procedure followed by GRAM in this case. The error message could display the faulty line and indicate the place where the error has occured.

This however cannot be done for the second kind of errors. They can be detected at the mapped level only during execution time. The function is then stored differently and there is no record, at this mapped level, of the form in which the function was entered unmapped.

However, this kind of error would have been detected at the unmapped level by an execution routine which could tell the nature nature of the error (rank or domain) and since we have at the mapped level a copy of the execution routine, it is still possible to deliver at this level an error message containing the following information.

a. The nature of the routine that has detected the error (addition or multiplication or iota operator routine etc...)

b. The nature of the error (rank error or domain error).

c. The values of the arguments for which the error was detected.

This means that the copies of the execution routines stored at the mapped level have to be slightly modified in their error message subroutines.

If the user is mapping functions already tested at the unmapped level and if he checks that all functions called by the one he is mapping have already been mapped before, there will therefore be no error message delivered during the mapping process; those are the functions referred to before as Syntax-error-free functions.

The Advantages Of The Mapped Level Suggestion

The Mapped Level modified APL has many of the advantages of a compiler while being quite distinct from it.

It is clear that the execution of the functions will be much faster at the mapped level. The fact that the syntax analysis has been done makes them close to compiled functions. However there is this important difference between the mapped level and a compiler: A compiler delivers an object program in the machine language that can be directly executed. In particular the compiled function should have all the needed instructions for storage handling, whereas a function stored at the mapped level is still in need of the module LINK at execution time.

It is also clear that the interractive feature of APL is not disrupted by the introduction of the mapped level as it would have been with the use of a compiler. In the case of most Fortran compilers for instance, alternating orders of compiling, executing, compiling, executing etc... require successive loadings of the compiler. In our case, the same interpreter will remain loaded in the computer while mapping or executing.

Another advantage is the flexibility of the combination of the two levels; in particular, it facilitates the editing and debugging process. A function can be tested and displayed at the unmapped level; the faulty line is then displayed with an indication of the place and the kind of error. It is then possible to execute parts of the line instead of executing the whole function. Such a facility would not have been available with a compiler. Once edited and debugged, the function may be stored at the mapped level.

ACKNOWLEDGEMENTS

The author is indebted to Dr. W.S. Adams, Dr. D.H. Bent and to Mr. G. Gabel for suggestions and fruitful discussions.

REFERENCES

1. In the Computing Center of the University of Alberta, a 360/67 IBM computer is used (mainly under M.T.S.). The c.p.u. time needed for loading an object program from a file is greater than the loading time needed in the APL case. There is therefore a class of programs that would take less time to be executed with APL than with a FORTRANG generated object program (if loading time is added to the execution time).

APL AS A NOTATION FOR STATISTICAL ANALYSIS

K. W. Smillie
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada

Abstract

This paper discusses the use of APL as a notation for statistical analysis and presents as a simple example the derivation of the chi-square statistic for independence in a two-way contingency table.

1. Introduction

The last few years have witnessed the remarkable growth in popularity of the APL language, until now it has been classified along with FORTRAN, PL/1, BASIC and a few other languages as one of the most important programming languages in use as present, and perhaps for the next decade. Such a development should indeed be most gratifying, especially to those who have been associated with the use of APL almost from the time of its first implementation and who must have had doubts from time to time about its survival. However, the acceptance of APL as a programming language has tended to obscure the origins of APL as an attempt to develop a notation for deriving and describing algorithms that was more powerful, more consistent and less ambiguous than conventional mathematical notation, and which was, incidentally, directly implementable on a computer. For these reasons it may be of some interest to consider the use and implications of APL as a notation. We shall consider as an example the derivation for a two-way contingency table of the maximum likelihood estimates of the expected frequencies on the assumption of the independence of the two categories of classification, and the use of these frequencies to obtain a convenient expression for the calculation of the chi-square statistic for independence. We shall first summarize the analysis in conventional notation and then derive the results rigorously in APL. We shall conclude with a few remarks on the use of APL as a notation.

2. Summary of analysis in conventional notation

Suppose that we have a two-way contingency table with r rows and c columns in which a sample of N observations is classified according to two attributes. Let f_{ij}, where $i=1,....,r$, and $j=1,....,c$, be the number of observations occurring in the ith class of the first category and the jth class of the second category. Let $r_i=\sum_j f_{ij}$ and $c_j=\sum_i f_{ij}$ be the marginal row and column totals, respectively. Thus $N=\sum_i r_i = \sum_j c_j$. If we let π_{ij} be the probability according to some hypothesis that an individual selected at random will fall in the ith class of the first category and the jth class of the second category, then the corresponding expected frequency is $e_{ij} = N\pi_{ij}$. A measure of the deviation of the observed frequencies from expectation is given by the statistic

$$\chi^2 = \sum_{i,j} \frac{(f_{ij} - e_{ij})^2}{e_{ij}}$$

which has the chi-square distribution with (r−1)(c−1) degrees of freedom.

If we assume that the two categories of classification are independent, then we may write $\pi_{ij}=\pi_i\pi_j$, where π_i is the marginal probability of an individual picked at random falling in the ith class of the first category independent of its classification according to the second category, and π_j is a similarly defined marginal probability for the jth class of the second category. Thus, in order to calculate the expected frequencies on the assumption of independence, we must estimate these marginal probabilities from the sample. According to the method of maximum likelihood the marginal probabilities are determined to maximize the likelihood $\prod_{i,j}(\pi_i\pi_j)^{f_{ij}}$, of the sample, where the π_i and π_j are subject to the restrictions $\sum_i\pi_i=1$ and $\sum_j\pi_j = 1$. Thus, we find the unrestricted maximum of the expression

$$L = \ell n \prod_{i,j} (\pi_i\pi_j)^{f_{ij}} + \lambda(\sum_i\pi_i-1) + \mu(\sum_j\pi_j-1),$$

where λ and μ are the Lagrangian multipliers. If we differentiate L partially with respect to π_i, π_j, λ and μ, set the partial derivatives to zero, and solve the resulting equations, we find that the estimates of π_i and π_j are given by $\hat{\pi}_i = r_i/N$ and $\hat{\pi}_j = c_j/N$, respectively. Thus, we find that the expected frequencies are given by $e_{ij} = r_ic_j/N$, and the value of X^2 may be simplified to

$$\chi^2 = N[\sum_{i,j} \frac{f_{ij}^2}{r_ic_j} -1].$$

3. Analysis in APL notation.

Suppose that we have a sample of observations arranged in a two-way contingency table according to two categories of classification, and that we wish to test the hypothesis that the two categories are independent. Let the data be represented by the two-dimensional array `F` so that `F[I;J]` , where `I∈ι(ρF)[1]` and `J∈ι(ρF)[2]` , represents the number of observations in the *Ith* class of the first category and the *Jth* class of the second category. For convenience, we shall let the row sums of `F` be given by the vector `R` , where

`[1] R←+/F` ,

the column sums by

`[2] C←+/[1]F` ,

and the total number of observations by

`[3] N←+/R` .

We shall assume that there is a probability matrix `P` , where `ρF ↔ ρF` , so that `P[I;J]` is the probability that an individual selected at random will fall in the *Ith* class of the first category and the *Jth* class of the second category. Since the expected frequencies in the contingency table are `N×P` , which may be represented by `E`, say, the deviation of the observed frequencies from expectation is given by the statement

`Z ↔ +/+/((F-N×P)*2)÷N×P`

Since we wish to test the hypothesis that the two categories are independent, we may replace `P` by the outer product `A∘.×B`, where `A ↔ +/P` gives the marginal probabilities of the first gategory regardless of the second category, and `B ↔ +/[1]P` gives the marginal probabilities for the second category. Since these marginal probabilities are unknown, they must be estimated from the sample data `F` . We shall derive these estimates by the method of maximum likelihood.

The likelihood of the observed sample is given by

`×/×/(A∘.×B)*F` ,

where `A` and `B` are subject to the restrictions `+/A ↔ 1` and `+/B ↔ 1`. If we take the natural logarithm of this expression and make use of some simple identities, we may write

`⍟×/×/(A∘.×B)*F ↔ +/+/F×⍟A∘.×B`

`↔ +/+/F×(⍟A)∘.+⍟B`

`↔ ((⍟A)+.×+/F)+(⍟B)+.×+/[1]F`

`↔ ((⍟A)+.×R)+(⍟B)+.×C` .

Therefore, we must find the unrestricted maximum of the expression

`L ↔ (((⍟A)+.×R)+(⍟B)+.×C)+(G×¯1++/A)+H×¯1++/B` ,

where `G` and `H` are the scalar Lagrangian multipliers.

Let us represent the maximum likelihood estimates of `A` and `B` by `AHAT` and `BHAT` , respectively. If we differentiate `L` with respect to `G` and set the derivative to zero, we have

`+/AHAT ↔ 1` .

Similarly, by differentiating `L` with respect to `H` we have

`+/BHAT ↔ 1` .

Now differentiate `L` with respect to the vector `A` and equate the derivative to zero, and obtain

`(÷AHAT)×R ↔ G` .

Therefore,

```
R ↔ G×AHAT .
```

If we sum both sides of this expression, we obtain

```
N ↔ G ,
```

and thus

```
AHAT ↔ R÷N .
```

Similarly, by differentiating L with respect to B we may show that

```
BHAT ↔ C÷N .
```

Thus the expected frequencies may be estimated by

```
E ↔ N×AHAT∘.×BHAT

  ↔ N×(R÷N)∘.×C÷N

  ↔ N×(R∘.×C)÷N*2

  ↔ (R∘.×C)÷N .
```

Therefore, the deviation of the observed frequencies from expectation is given by

```
Z ↔ +/+/((F-E)*2)÷E

  ↔ +/+/((F×F)-(2×F×E)-E×E)÷E

  ↔ +/+/(F×F÷E)-(2×F)-E

  ↔ (+/+/F×F÷E)-(2×+/+/F)-+/+/E .
```

Now

```
+/+/F ↔ N ,
```

and

```
+/+/E ↔ +/+/(R∘.×C)÷N

      ↔ ((+/R)∘.×+/C)÷N

      ↔ N×N÷N

      ↔ N .
```

Therefore,

```
Z ↔ (+/+/F×F÷E)-(2×N)-N

  ↔ (+/+/F×F÷E)-N

  ↔ (+/+/F×F÷(R∘.×C)÷N)-N

  ↔ N×(+/+/F×F÷R∘.×C)-1

  ↔ N×¯1++/+/F×F÷R∘.×C .
```

Therefore, we may compute the test statistic for the deviation of the observed frequencies from expectation by the statement

```
[4]  Z ↔ N×¯1++/+/F×F÷R∘.×C
```

4. Implementation

The four numbered statements appearing in the analysis of the preceding section may be considered to be the body of the monadic defined function *CHSQ* with a right argument *F* and a result *Z*. This function is given in Figure 1, which also gives two examples of its use with some sample data *F1* and *F2*.

```
    ∇ Z←CHSQ F;C;N;R
[1]   R←+/F
[2]   C←+/[1] F
[3]   N←+/R
[4]   Z←N×¯1++/+/F×F÷R∘.×C
    ∇


          F1
   5   9
  11  15

          CHSQ F1
0.1648351648


          F2
  42  31  12  17
  34  25  31  22
  48  37  18  13

          CHSQ F2
15.27832566
```

Figure 1. Function *CHSQ* and some examples of its use.

5. Conclusions

The example which we have discussed in this paper is an illustration of how the use of APL as a notation may remove the need for programming in the conventional sense since selected statements of the analysis become the body of a defined function which is executed on the computer. Although this example is a very simple one, and, indeed, was chosen for this reason, the ease with which APL was used for the analysis hopefully will suggest that such an approach may be worth considering for other more complicated problems. Some topics which come immediately to mind are multiple regression analysis, analysis of variance for factorial designs, nonparametric methods, and the analysis of incomplete block designs. The limited work which appears to have been done on some of these topics is most encouraging, and suggests that most interesting results await the persons who will consider them in detail. Only by gaining experience in the use of APL as a notation, as well as a programming language, will the adequacy of APL in this role, as well as the direction of further extensions to the language, become apparent. It is hoped that this short paper may help stimulate research on these subjects.

6. Reference

Keeping, E.S., 1962. Introduction to Statistical Inference. D. van Nostrand Co., Inc., Princeton, New Jersey.

AN ADAPTIVE QUERY SYSTEM

E. Kellerman
IBM Corporation
Systems Development Division
P. O. Box 6
Endicott, New York 13670

This paper will describe an adaptive query program coded in APL. The purpose of the program is to allow users to ask questions in everyday English and to receive answers with minimal delay.

The program is "taught" the correct answers by a human "instructor", sitting at a terminal, asking it questions on the subject of interest. As the program learns the answers to some of the questions, it attempts to guess at the answer to "similar" questions. If the program is "very" sure of the answer it has given, then it does not request verification, otherwise it requests confirmation of the correctness of the answer it has given. Even when the program does not request verification of the answer given, the instructor still has the option of informing the program that an incorrect reply was given, by entering '?'. If the program gives an incorrect reply, but the instructor feels that the program should know the answer, he can request the program to 'try again'. Consider the following terminal session, which starts with a complete unknowledgeable program. The lines typed by the terminal are preceded by '***':

```
HOW DO I GET OFF?
***WHAT IS ANS ?
HIT CARRIAGE RETURN!!

WHAT PROGRAMS ARE AVAILABLE?
***WHAT IS ANS ?
LIST, DUP, AND INTER.

WHAT DOES LIST DO?
***WHAT IS ANS ?
LISTS CARDS.

WHAT DOES DUP DO?
***LISTS CARDS.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
DUPLICATES CARDS.

WHAT DOES INTER DO?
***LISTS CARDS.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
INTERPRET CARDS.

HOW DO I LIST CARDS?
***WHAT IS ANS ?
USE LIST.

HOW DO I DUPLICATE CARDS?
***USE LIST.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
USE DUP.

HOW DO I INTERPRET CARDS?
***USE DUP.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
USE INTER.

HOW DO I DUP CARDS?
***USE LIST.
*** WAS THAT RIGHT?
TRY AGAIN
***USE DUP
*** WAS THAT RIGHT?
YES

HOW DO I DUP CARDS?
***USE DUP
*** WAS THAT RIGHT?
YEP

HOW DO I INTER CARDS
***USE LIST.

?
*** WAS THAT RIGHT?
TRY AGAIN
***USE INTER.
*** WAS THAT RIGHT?
YEAH

HOW DO I INTER. CRDS?
***USE INTER.

HOW DO I LIST CDS.
***USE LIST.

WHAT PROGRAMX AVAILABLE?
***LIST, DUP, AND INTER.

WHAT DOS DUP DO?
***DUPLICATES CARDS.

HOW DO I DUPLIMATE CRDS.?
***USE DUP.

HOW DO I INTERPRETARE CRDS.?
***USE INTER.

WHAT DO INTER DO
***INTERPRET CARDS.
```

```
HOW DO I LST CRDS?
***USE LIST.

WHAT DOES LIST DO????
***LISTS CARDS.

WHAT DEOS INTRE DO?
***INTERPRET CARDS.

HOW DO I SIGN OFF.
***HIT CARRIAGE RETURN!!

HOW DO I GET OFF?
***WHAT IS ANS ?
HIT CARRIAGE RETURN!!

WHAT PROGRAMS ARE AVAILABLE?
***WHAT IS ANS ?
LIST, DUP, AND INTER.

WHAT DOES LIST DO?
***WHAT IS ANS ?
LISTS CARD.

WHAT DOES DUP DO?
***LISTS CARDS.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
DUPLICATES CARDS.

WHAT DOES INTER DO?
***LISTS CARDS.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
INTERPRET CARDS.

HOW DO I LIST CARDS?
***WHAT IS ANS ?
USE LIST.

HOW DO I DUPLICATE CARDS?
***USE LIST.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
USE DUP.

HOW DO I INTERPRET CARDS?
***USE DUP.
*** WAS THAT RIGHT?
NO
***WHAT IS ANS ?
USE INTER.

HOW DO I DUP CARDS?
***USE LIST.
*** WAS THAT RIGHT?
TRY AGAIN
***USE DUP.
*** WAS THAT RIGHT?
YES

HOW DO I DUP CARDS?
***USE DUP.
*** WAS THAT RIGHT?
YEP

HOW DO I INTER CARDS
***USE LIST.

?
*** WAS THAT RIGHT?
TRY AGAIN
***USE INTER.
*** WAS THAT RIGHT?
YEAH

HOW DO I INTER. CRDS?
***USE INTER.

HOW DO I LIST CDS.
***USE LIST.

WHAT PROGRAMX AVAILABLE?
***LIST, DUP, AND INTER.

WHAT DOS DUP DO?
***DUPLICATES CARDS.

HOW DO I DUPLIMATE CRDS.?
***USE DUP.

HOW DO I INTERPRETARE CRDS.?
***USE INTER.

WHAT DO INTER DO
***INTERPRET CARDS.

HOW DO I LST CRDS?
***USE LIST.

WHAT DOES LIST DO????
***LISTS CARDS.

WHAT DEOS INTRE DO?
***INTERPRET CARDS.

HOW DO I SIGN OFF.
***HIT CARRIAGE RETURN!!
```

At the heart of the adaptive query program is an algorithm for evaluating the similarity of two character strings. This algorithm was developed by G. L. Rouse, D. C. Gause and the author. An application for a patent has been made. A description of the algorithm is now given. Call the character strings to be compared A and B. Then:

Step 1:

Form a matrix, M, by assigning a 1 to M[I;J] if and only if A[I] = B[J]. Otherwise M[I;J] is set to 0; This matrix is formed by the following APL expression: M←A ∘ . = B. For example, if A 'ANNE' and B 'ANNIE' then M is:

	A	N	N	E
A	1	0	0	0
N	0	1	1	0
N	0	1	1	0
I	0	0	0	0
E	0	0	0	1

Step 2:

If a row or a column of M contains more than one 1, then retain only the one closest to the main diagonal; the following APL expression does this:

```
M←N=⍉((ρB),ρA)ρ(×/ρM)ρSS+0=SS←⌈/(N←M×1000-|(ιρA)∘.-ιρB)
```

Note that if two 1's are equidistant from the diagonal, the expression would retain both.

From the preceding example we would get:

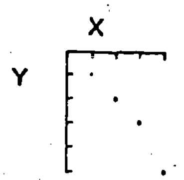
M =	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	0	0
	0	0	0	1

Step 3:

Consider the 1's in M as points on an X-Y coordinate system. That is, if M[I;J] is equal to 1 then we have a point with the Y-coordinate equal to I, and the X-ccoordinate equal to J. The APL expression for this is:

```
X←D/(S←ρD←,M)ριρB
Y←D/,⍉((ρB),ρA)ρSριρA
```

From the preceding example we would get:



Step 4:

The standard correlation coefficient (which measures linear dependence) of the points is taken as a measure of similarity between the two strings. The closer to 1, the greater the similarity, the closer to -1 the greater the difference. The following APL expression evaluates the standard correlation coefficient:

```
CC←((N×+/X×Y)-X1×Y1)÷(((N×+/Y*2)-(Y1←+/Y)*2)*0.5)×((N×+/X*2)-(X1←+/X)*2)*0.5
```

For example, consider the results of applying this algorithm to determine the similarity of the question *'WHAT IS TODAY?'* with several other phrases:

Phrase	Correlation Value
WHAT IS TODAY	1.00
WHTA IS TODAY	0.994
WHAT IS TODAY	0.997
WHAT IS TDAY?	0.997
WHAT TODAY	0.97
WHAT DAY	0.92
TODAY IS WHAT	-0.05
YATOD SI TAHW	-0.36
MY NAME IS ED	0.001

With this algorithm in hand, the implementation of the Query System is fairly straightforward. A table is kept of questions seen and their associated answers. Associated with each question is a threshold which the correlation value must exceed in order for verification not to be requested. This verification threshold is adjusted so that verification is not requested more than once for any given input question. Also associated with each question is a threshold value which the correlation value must exceed for the question to be considered a match. When a new question is entered, a correlation value between that question, and all the questions in the table is computed. Only questions whose correlation value is higher than the associated threshold value are considered as candidates. Amongst the candidates, the one with the highest correlation value is chosen, and the answer associated with that question is given. If the answer given turns out to be incorrect, the threshold value associated with the selected question is raised to be slightly higher than the correlation value obtained for that question, thus insuring that the question would not be a candidate when the same question is posed to the sytem. Also, the new question is stored and the program asks asks what the correct answer to it should be, thus, another entry into the question-answer table is made. The threshold initially associated with a new question is set to a "low value."

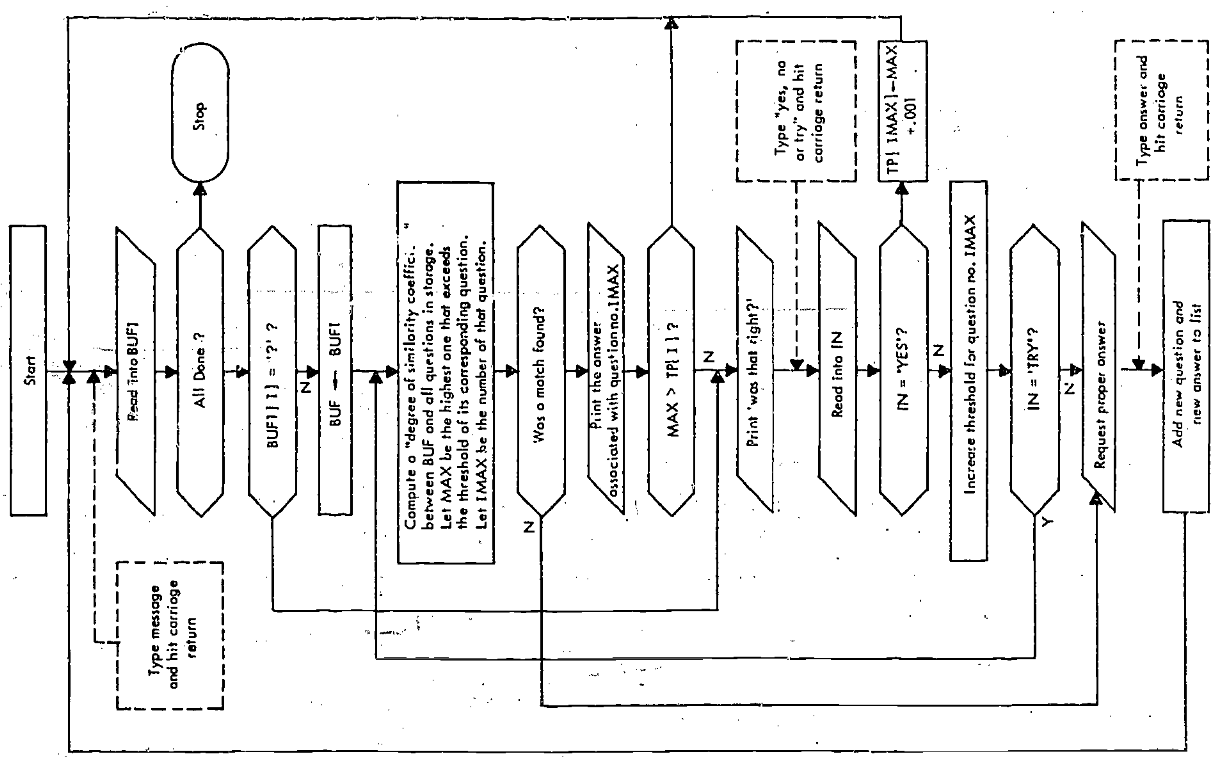
The attached flowchart gives a more detailed description of the program.

Note that this adaptive query system has many applications; some possible uses include:

- allowing CAI (computer aided instruction) users to ask questions, at any point, about the subject being taught,

- questioning a system (such as APL) to find the type and use of available commands, and

- ignoring simple spelling errors in compilers.



MICROPROGRAM TRAINING - AN APL APPLICATION

Ray Polivka and Kent Haralson
IBM Corporation
Poughkeepsie, New York 12601

Introduction: Nature of Microprogramming

When given a computer system, probably the first thing a user looks at is the instruction set. This information is usually found in a Principles of Operation manual. Here also resides the architectural flavor of the system. Now move from the user of a computer system to the implementors of a computer system. At what do they look? Certainly they must understand the instruction set and architecture. But, in addition, they need to know the nature of the hardware with which they have to work. They must know such things as the functions which can be executed by the Arithmetic and Logic Unit and how data can flow between the storage registers that make up the hardware of the computer. All of this information in great detail is found in the functional specification of the computer system. Much of this information is represented graphically as a data flow. It describes how data can move within the hardware that comprises the computer system.

The computer system can not yet operate since a very important item, the element of time, is missing. The determination of when data should move within the data flow makes up what is referred to as control design. The specification of these controls for a data flow is a very intricate and involved affair. Microprogramming is one technique of control design. One of its salient features is that all the control information is stored in an orderly fashion as an array. This array is referred to as a control store. Information is selected from control store a small portion at a time. This portion, called a microinstruction, contains the information necessary to control the data flow for a small period of time, usually a machine cycle.

Objective

The use of microprogramming has grown quite rapidly over the past few years. This in turn has produced a corresponding increase in interest in microprogramming. SDD Programming Education has developed a course in microprogramming which presents to a student the concepts and fundamental principles underlying microprogramming. In addition to such concepts and principles, it is highly desirable that the student actually do some microprogramming. This requirement was met by developing an interactive APL simulation package. This package is based upon a hypothetical machine developed by C. W. Gear in his textbook Computer Organization and Programming. Here we have a well defined architecture with a data flow large enough to be realistic and yet simple enough to avoid unnecessary confusion over details. With this package the student is able to develop and execute both machine and micro code. The APL in which it was developed is transparent to the student.

This package is used in conjunction with printed material presented during the course. It consists of (1) A Principles of Operation manual (6 pages), (2) A User's Manual (7 pages), and (3) A Microprogramming Manual (25 pages). The Principles of Operation manual describes the architecture of the machine and its instruction set. The User's manual defines the nature of the assembly language. Finally, the Microprogramming manual contains the data flow and accompanying descriptive material as well as the microprogramming language in which to write the micro-code.

Usage

No knowledge of APL is necessary. Initially the student need type only three keywords, IPL, TEST1, and START, to have a complete simulation of the execution of three machine instructions and its supporting microcode. In this way the student can become familiar with the mechanics of the package. Figure 1 contains the data flow which is simulated. Figure 2, illustrates the sequence of events that occur when the three keywords are entered. When he has familiariazed himself with the procedure, he has available another package TEST2 which he may use. This package loads memory with a small assembly program, but he must provide the supporting microprogram. The nature of TEST1 and TEST2 are described in EXPLAIN1 and EXPLAIN2, respectively. From this point he should be able to generate both assembly instructions and the supporting microcode. Figure 3 portrays a sequence of assembly instructions as entered on the terminal. Figure 4 illustrates the input of a set of microinstructions as well as three assembly instructions. Note that it also illustrates some of the diagnostics that the user can get. The facilities available to the student can be subdivided into three parts. The first part consists of the functions which simulate the machine definition. The second part consists of those functions which initialize or reset memory and control store. The third part consists of extensive diagnostic aids. With these aids he has the ability to dump portions of both control

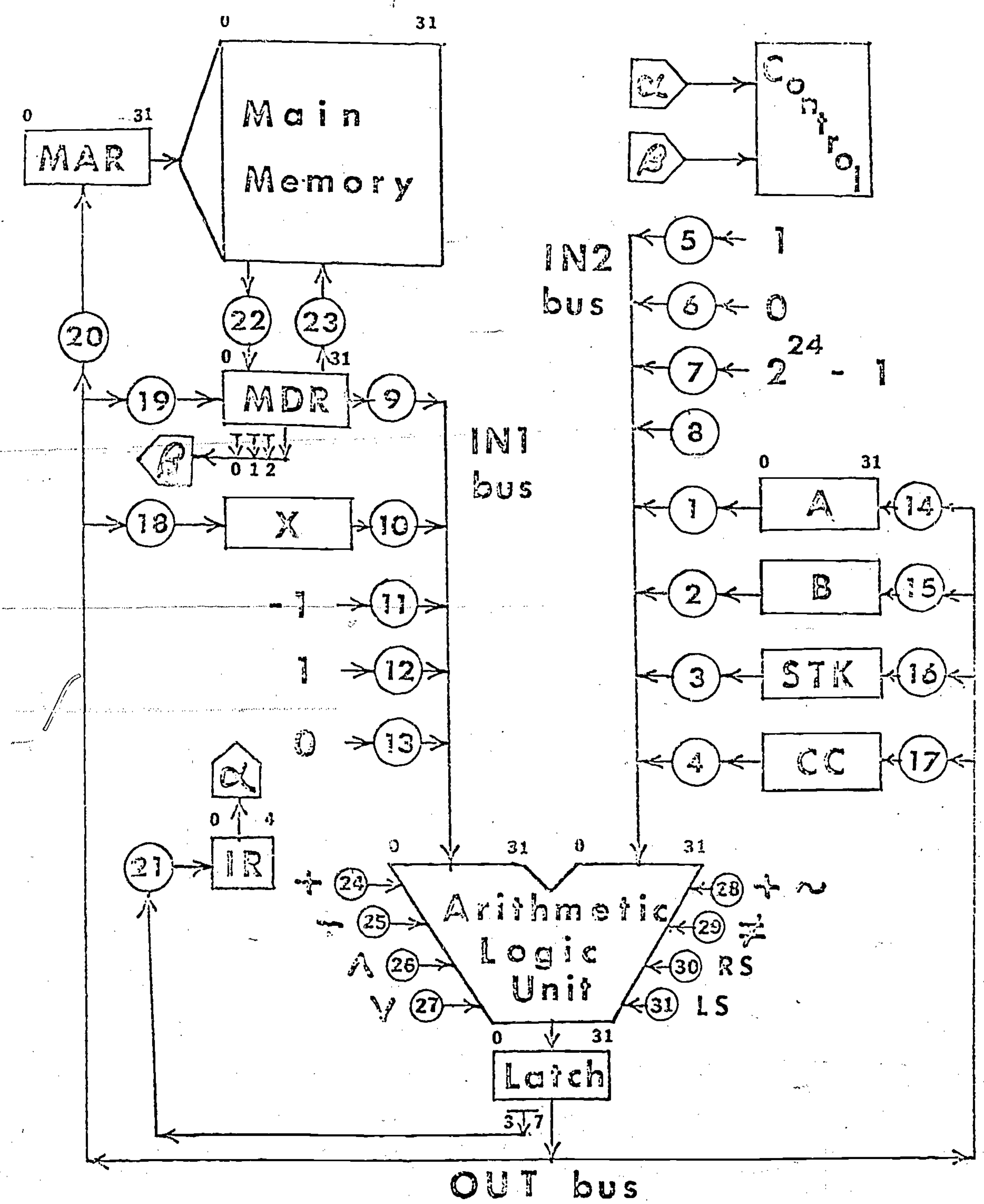


Figure 1 Data Flow

```
      IPL
SYSTEM RESET IS COMPLETE

      TEST1

      START
WHAT IS THE STARTING POINT IN MAIN MEMORY (BASE 1 INDEXING)
():
      1

     MAR= 0    MDR= 0    A= 0    B= 0
     X= 0    STK= 101    CC= 1    IR= 0
     0 MACHINE CYCLES USED

M-INSTR EXECUTED
1  ADDM   ZERO,CC   ,MAR,R

     MAR= 1    MDR= 4    A= 0    B= 0
     X= 0    STK= 101    CC= 1    IR= 0
     1 MACHINE CYCLES USED

M-INSTR EXECUTED
2  ADDM   ONE  ,CC   ,CC  ,P

     MAR= 1    MDR= 4    A= 0    B= 0
     X= 0    STK= 101    CC= 2    IR= 0
     2 MACHINE CYCLES USED

M-INSTR EXECUTED
3  ADDM   MDR  ,ZERO,IR  ,P
          .
          .
          .
          .
          .
          .
          .
          .
          .
     MAR= 3    MDR= 738197504    A= 0    B= 5
     X= 0    STK= 99    CC= 4    IR= 12
     31 MACHINE CYCLES USED

M-INSTR EXECUTED
44  TRM   13
END OF JOB ********    TIME USED = 32 MACHINE CYCLES
```

Figure 2

```
 1   READ       0  0  14
 2   READ       0  0  15
 3   LOAD       0  0  13
 4   LOAD       0  0  15
 5   LOAD       0  0  14
 6   LOAD       0  0  14
 7   ADD        0
 8   SUB        0
 9   ADD        0
10   STORE      0  0  16
11   WRITE      0  0  16
12   STOP       0
13   CONSTANT   2
14   CONSTANT   0
15   CONSTANT   0
```

Figure 3

```
      IPL
SYSTEM RESET IS COMPLETE

      1    ADDM    ZERO,CC,MAR,R
      2    ADDM    ONE,CC,CC,P
      3    ADDM    MDR,ZERO,IR

*** INCORRECT NUMBER OF FIELDS
      3    ADDM    MDR,ZERO,IR,P
      4    T2      6
      5    TI      32
      6    ANDM    MDR,MASK,B,P
      7    T1      9
      8    ADDM    X,B,B,P
      9    T0      12
     10    ADDM    ZERO,B,MSR,R
VALUE ERROR
     10 ADDM ZERO,B,MSR,R
                     ^
     10    ADDM    ZERO,B,MAR,R
     11    TRM     6
     12    TI      64
     44    TRM     148
     64    TRM     95
     95    ADDM    ZERO,B,MAR,R
     96    ADDM    NONE,STK,STK,P
     97    ADDM    ZERO,STK,MAR,M
     98    TRM     1
    148    STOPM   0

      1    LOAD      0  0  3
      2    STOP      0
      3    CONSTANT  7
```

Figure 4

```
     SETTRACE
ENTER CTL STORE LOCS. TO BE TRACED
  'ALL' OR 'NONE' ARE ACCEPTABLE
[]:
     NONE

     START
WHAT IS THE STARTING POINT IN MAIN MEMORY (BASE 1 INDEXING)
[]:
     1
END OF JOB ********   TIME USED = 19 MACHINE CYCLES



     SETTRACE
ENTER CTL STORE LOCS. TO BE TRACED
  'ALL' OR 'NONE' ARE ACCEPTABLE
[]:
     ALL
INDICATE NATURE OF TRACING
    0 : CTL ST LOC, MNEM, REGS
    1 : CTL ST LOC, MNEM
    2 : CTL ST LOC
[]:
     2

     START
WHAT IS THE STARTING POINT IN MAIN MEMORY (BASE 1 INDEXING)
[]:
     1
1
2
3
4
6
7
9
12
64
95
96
97
98
1
2
3
4
5
44
148
END OF JOB ********   TIME USED = 19 MACHINE CYCLES
```

Figure 5

```
     HELP

PROCESSING CNTRL INSTR. 148  AND MEMORY LOCATION 3
REGISTER CONTENTS ARE

     MAR= 2    MDR= 778197504    A= 0    B= 3
     X= 0    STK= 100    CC= 3    IR= 12
     19 MACHINE CYCLES USED

THE CNTRL INSTRUCTION IS:
          0000000001010000

THE CURRENT WORD IN MEMORY YOU ARE PROCESSING IS:
          00000000000000000000000000000111

THE TOP THE STACK IS:
          00000000000000000000000000000111


     DISPLAY MCODE
STARTING POINT
[]:
     1
STOPPING POINT
[]:
     3

1  ADDM    ZERO,CC   ,MAR,R
2  ADDM    ONE ,CC   ,CC ,
3  ADDM    MDR ,ZERO,IR ,


     DISPLAY MEMORY
STARTING POINT
[]:
     3
STOPPING POINT
[]:
     3

     BINARY NOTATION                                 HEXADECIMAL NOTATION
0 0 000000 000000000000000000000000111               00000007
```

Figure 6

memory and main memory and to selectively trace the execution. Figure 2 illustrates an execution with a full trace. Figure 5 shows the nature of trace selection. In Figure 6 we see two more diagnostic aids, HELP and DISPLAY. HELP yields a snapshot of the pertinent parts of the data flow. It is useful when the microprogram goes awry. DISPLAY allows one to display either main memory or control store. Control store is displayed mnemonically and main memory is displayed in binary and hexadecimal notation.

Techniques Used

This package was developed using the APL/360 XM6 release; it resides completely within a 32K workspace. The key function is the START function. It is the embodiment of the given data flow. The timing of events is incorporated in the sequence that the actions are performed in it. The function is written deliberately in a vertical fashion. This was done in order that the changes that a student might like to make to the data flow could be accomplished easily. The START function steps successively through a binary array called MCODE. MCODE is the control store in which each row is a microinstruction. The sequence of micoinstructions chosen from MCODE is controlled by another binary array MEMORY. In MEMORY resides the binary representation of the machine instructions. Again each row contains one machine instruction.

Building these binary matrices is no more palatable than writing programs in binary. A standard assembler approach provides a mnemonic means of creating the proper bits patterns. Thus each machine instruction is implemented as a dyadic function. The name of it matches the name of the instruction. Thus for

3 LOAD 0 0 15

which is a machine instruction, LOAD is, in APL, a dyadic function. Its right argument is a three component vector defining the nature of the effective address. The first and second components are binary and define whether indirect addressing or indexing is to occur. The third component is an absolute address component. The left argument is the absolute address in MEMORY which will contain the binary configuration generated by the mnemonic instruction.

The microinstructions for this machine are in either of two forms, the data flow controlling form and the sequence controlling form, see figure 7. For the data flow controlling microinstructions each possible function permitted in the function field is implemented as a dyadic function. Its left argument is again the absolute address in MCODE which will contain the generated bit configuration. Its right argument is a four component vector which corresponds to the other four fields in the microinstruction. The general form is

Loc FCN In1, In2, Output, Memory

All the components of the right argument may be written mnemonically. For example, to add the contents of register X to the contents of register B, put the result in register MAR, and finally issue a read of memory would appear as

7 ADDM X,B,MAR,R

The sequence controlling microinstruction is also a dyadic function. The name of the function matches the mnemonic for the test condition. Its left argument is the absolute address in MCODE for the generated bit configuration. Its right argument is the absolute address in MCODE of the next microinstruction to be in control. For example the microinstruction

7 TRM 64

when executed causes an unconditional transfer to the microinstruction in control store location 64.

This technique while not elaborate proved quite adequate. If an error were made in either an instruction or a microinstruction, correction simply consists of reentering it. Occasionally this was hard to grasp by an experienced programmer since he expected something more involved. If the user is a knowledgable APL user, he may use the defined function facility of APL as an assembler. Instead of entering the instruction individually, they may be collected together as a niladic function. For example,

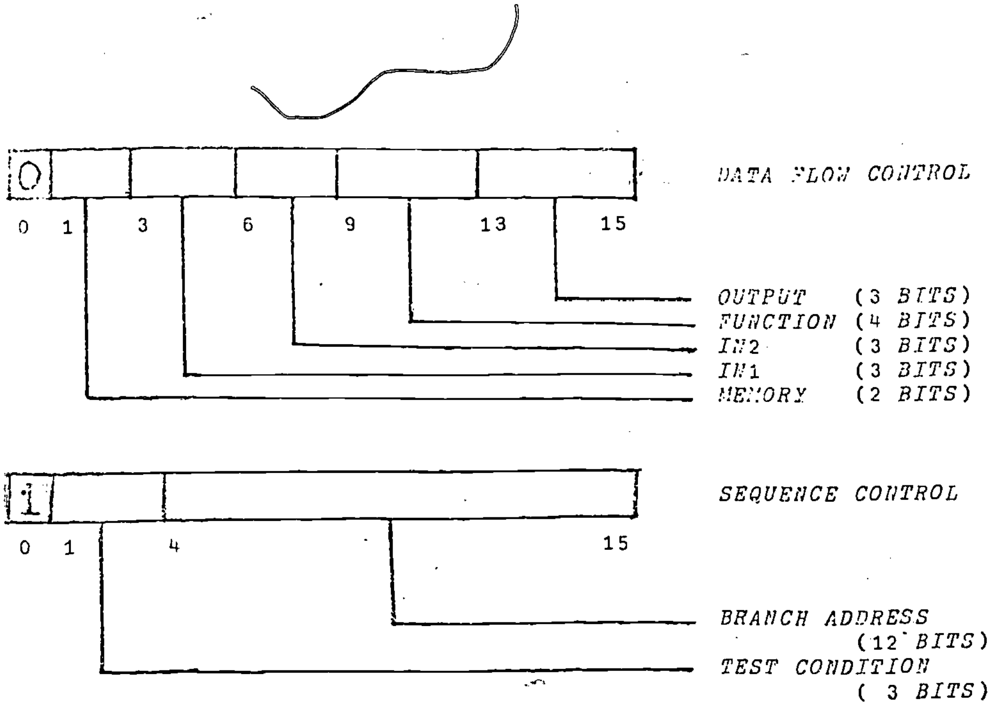


Figure 7

		FULL TRACE	PARTIAL TR.*	NO TRACE
1. 3 MACHINE INSTR. 32 CNTRL INSTR. (RUN ON CP-67)	CLOCK CPU	16 MIN 35 SEC 17.25 SEC	1 MIN 45 SEC 5.5 SEC	29 SEC 4.25 SEC
2. 42 MACHINE INSTR. 558 CNTRL INSTR. (RUN ON CP-67)	CLOCK CPU		28 MIN 52 SEC 1 MIN 35 SEC	2 MIN 33 SEC 1 MIN 6 SEC
3. 3 MACHINE INSTR. 32 CNTRL INSTR. (RUN ON MOD 85)	CLOCK CPU	7 MIN 55 SEC 1.5 SEC	30.5 SEC .917 SEC	20.25 SEC .883 SEC

* JUST CONTROL STORE LOCATIONS DISPLAYED

Figure 8

```
∇EXAMPLE
[1]   3  LOAD  0 1 7
[2]   4  STORE 0 0 39
     ∇
```

Changes to programs then can be made via the editing facilities of APL. A word of caution though, this does tend to fill up the workspace.

Timing and Effort

This training package was essentially written in about 100 man hours over a period of three weeks. Many improvements were suggested and even made by students as they used it. This proved to be very valuable input.

Since several levels of simulation are involved, the execution time for the individual machine instructions are inherently slow. Figure 8, contains some sample times. The first and third cases exercise the microcode to execute two LOADs and a STOP. The second case exercises all the microcode to execute all but one of the machine instructions at least once.

Projections

This package can serve as a ready foundation for further modifications and extensions. For example several additional instructions could be added to the repertoire of the machine. Or one could make changes in the data flow. The addition or deletion of registers and modifications to the data paths are possibilities. The timing with respect to memory references could be made more realistic. A more ambitious undertaking would be to treat this data flow or subset as an I/O control unit. With two such versions, one an I/O control unit and one a CPU, dynamic interaction could occur.

Summary

This package has been successfully used to introduce the concept of microprogramming. It enables a student to actually write and execute microcode in an interactive environment. This leads to a proper appreciation of what is involved. The package appears readily extendable offering several avenues to follow.

A technical report containing complete student manuals is currently being written and will be available from the authors.

REFERENCE

1. Gear, C. W., Computer Organization and Programming, McGraw-Hill, 1969.

ECAPL

AN APL ELECTRONIC CIRCUIT ANALYSIS PROGRAM

Randall W. Jensen, Jerry A. Higbee and Paul M. Hansen
Electronic Design Associates
1536 East 1220 North
Logan, Utah 21

Management Systems Corporation
15 North West Temple
Salt Lake City, Utah 84103

ECAPL (APL Electronic Circuit Analysis Program) is an interactive integrated system of programs developed by Randall W. Jensen, Terry A. Higbee, and Paul M. Hansen of Electronic Design Associates for Management Systems Corporation, 15 North West Temple, Salt Lake City, Utah 84103. The programming system was developed primarily to aid the electrical engineer in the design and analysis of electronic circuits. The system's capabilities are similar to those of the IBM/360 ECAP[1] program, however, the techniques used to perform the analysis are different. The user familiar with ECAP will have little difficulty making the transition to ECAPL.

The ECAPL system consists of four clsely related programs.

Input Language: This program acts as the communication link between the user and the three analysis programs. The interactive language is user-oriented and allows complex circuits to be simply described to the computer. The four basic types of statements used in the program completely define the topology of the circuit, the circuit element values, the type of analysis to be performed, the driving functions, and the output required. Each input statement is analyzed when entered for validity and syntax so that the user is completely isolated from the APL system. Comprehensive diagnostics are supplied to aid input debugging. These features make it possible to learn to use ECAPL in a short time.

DC Analysis: The dc analysis program obtains the dc or steady-state solutions of linear electrical networks and provides the worst-case analysis, standard deviation (statistical) analysis, and sensitivity coefficients if requested. This program also provides an automatic parameter and topology-modification capability.

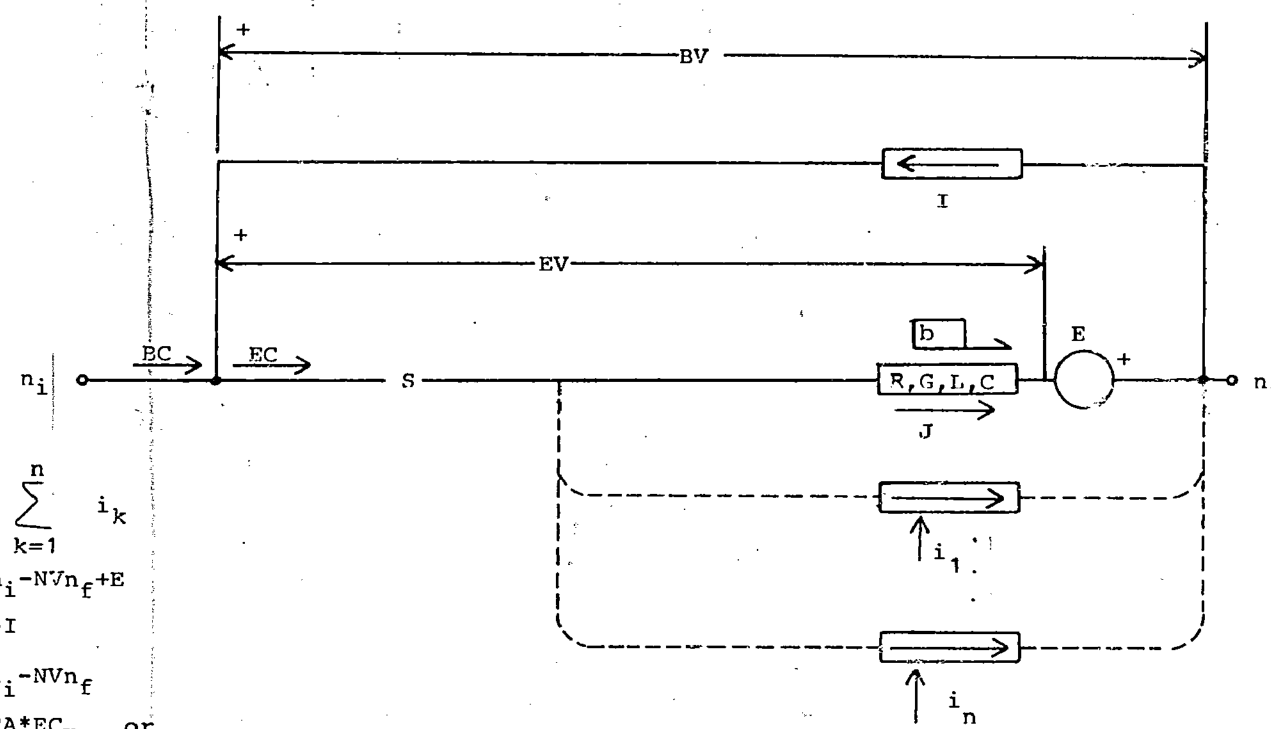
AC Analysis: The ac analysis program obtains the steady-state solution of electrical networks subject to sine-wave excitation at an arbitrary fixed frequency. Since this program also contains the automatic network-modification capability, it is easy to obtain frequency and phase-response solutions.

Transient Analysis: The transient analysis program provides the time-response solution of linear or nonlinear electrical networks subject to arbitrary, user-specified driving functions. Nonlinear elements are modeled by using combinations of switches and resistors, capacitors, or inductors to provide piecewise linear approximations to the nonlinear characteristics.

ECAPL is not difficult to use. It is not necessary for the user to have any knowledge of APL or of the internal mechanics of the ECAPL system and he needs no prior computer or programming experience. However, he must know the techniques of communicating with the computer via ECAPL. These include (1) the means of converting the circuit schematic to a written format acceptable to the program, (2) the information required to obtain the desired analysis, and (3) the knowledge to interpret the output results. The techniques involved in using ECAPL are easily learned by an electrical engineer because ECAPL's input language and its output are written in electrical circuit terminology.

The ECAPL input data provides information describing the interconnection of the branches, the types of elements, their values, tolerances, gains, inductive couplings, initial conditions, and dynamic changes in their values. There are only eight different data card types required to provide thiis information: passive branch data, current gain or transconductance, mutual inductive coupling, switches to provide dynamic changes, independent voltage and current sources, and three types of time-dependent sources. The eight types of data cards specify the element values in a "standard" branch.

The standard branch shown in Figure 1 is the basic building block of ECAPL. It consists of a nonzero passive element; R, G, L, or C. In addition, it may include a voltage source E and/or a switch A in series with the element and/or an independent current source I in parallel. It may also contain any number (200) of dependent current sources i, as shown, in parallel with the element. The positive current directions and voltage polarities are shown in the figure.



$$EC = J + \sum_{k=1}^{n} i_k$$

$$EV = NVn_i - NVn_f + E$$

$$BC = EC - I$$

$$BV = NVn_i - NVn_f$$

$$i_n = BETA*EC_m \quad \text{or}$$

$$GM*EV_m$$

where m = 'from' branch

Figure 1

Besides the branch data statements, ECAPL also includes two additional types of input statements; command and control. The command statement EX or EXECUTE signals the end of the input data and causes the analysis to begin.

Solution control statements are of two types. The first contains data of a general nature that pertains to the analysis of a circuit(e.g., frequency, time step, etc.). The second type specifies calculations to be made, other than a nominal solution (e.g., sensitivity coefficients, worst case, etc.). The page control (PC) output control statement is a special form of solution control statement which stops the analysis after each block of output to allow the user to insert a new sheet of paper in the terminal. The analysis is reactivated by depressing the carriage return key.

The statement formats for dc analysis, ac analysis and transient analysis are summarized in Figure 2.

STATEMENT TYPE		STATEMENT FORM	DEFAULT VALUE
COMMAND		EX [ECUTE]	
DATA	PASSIVE	$\left\{\begin{matrix} R \\ G \\ L \\ C \end{matrix}\right\}$ $b, n_i \rightarrow n_f = P_1$	
	DEPENDENT CURRENT SOURCE	$\left\{\begin{matrix} B[ETA] \\ GM \end{matrix}\right\}$ $n, b_f \rightarrow b_t = P_1$	
	INDEPENDENT SOURCE	$\left\{\begin{matrix} E \\ I \end{matrix}\right\}$ $b = P_1$	
SOLUTION CONTROL		WO [RST CASE], P_6 SE [NSITIVITIES] ST [ANDARD DEVIATIONS] TE [RROR] $= P_4$ SH [ORT] $= P_4$ OP [EN] $= P_4$	 0.001 0.01 10^7
OUTPUT CONTROL		PC	

Figure 2(a) DC Analysis Statement Formats

STATEMENT TYPE		STATEMENT FORM	DEFAULT VALUE
COMMAND		EX[ECUTE]	
DATA	PASSIVE	$\left\{\begin{matrix} R \\ G \\ L \\ C \end{matrix}\right\}$ $b, n_i \to n_f = P_2$	
	DEPENDENT CURRENT SOURCE	$\left\{\begin{matrix} B[ETA] \\ GM \end{matrix}\right\}$ $n, b_f \to b_t = P_2$	
	INDUCTIVE COUPLING	$M, b_1 \to b_2 = P_2$	
	INDEPENDENT SOURCE	$\left\{\begin{matrix} E \\ I \end{matrix}\right\}$ $b = P_3$	
SOLUTION CONTROL		$1E[RROR] = P_4$	0.001
OUTPUT CONTROL		PC	

Figure 2(b) AC Analysis Statement Formats

STATEMENT TYPE		STATEMENT FORM	DEFAULT VALUE
COMMAND		EX[ECUTE]	
DATA	PASSIVE	$\left\{\begin{matrix} R \\ G \\ L \\ C \end{matrix}\right\}$ $b,n_i \rightarrow n_f = P_5$	
	DEPENDENT CURRENT SOURCE	$\left\{\begin{matrix} B[ETA] \\ GM \end{matrix}\right\}$ $n,b_f \rightarrow b_t = P_5$	
	SWITCH	SWn,b $\left\{\begin{matrix} \uparrow \\ \downarrow \end{matrix}\right\}$ P_6	
	DC SOURCES & INITIAL CONDITIONS	$\left\{\begin{matrix} E \\ I \end{matrix}\right\}$ $b=P_4$	
	TIME-DEPENDENT SOURCES		
	Nonperiodic	$\left\{\begin{matrix} E \\ I \end{matrix}\right\}$ Tb,P_6	
	Periodic	$\left\{\begin{matrix} E \\ I \end{matrix}\right\}$ Pb,P_7	
	Sinusoidal	$\left\{\begin{matrix} E \\ I \end{matrix}\right\}$ Sb,period,peak,dc,to	

Figure 2(c) Transient Analysis Statement Formats (1/2)

STATEMENT TYPE	STATEMENT FORM	DEFAULT VALUE
SOLUTION CONTROL	EQ[UILLIBRIUM]	
	TI[ME STEP]=P_4	1.0
	1E[RROR]=P_4	0.001
	2E[RROR]=P_4	0.001
	OU[TPUT INTERVAL]=P_4	1
	IN[ITIAL TIME]=P_4	0.0
	FI[NAL TIME]=P_4	0.0
	SH[ORT]=P_4	0.01
	OP[EN]=P_4	10^7
	CO[NTINUE]	
	SO[URCE INCREMENT]=P_4	1
OUTPUT CONTROL	PC	

Figure 2(d) Transient Analysis Statement Formats (2/2)

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Value	[1] ✓	✓		✓	✓	✓	
Nominal (decimal tolerance)	✓						
Nominal (min, max)	✓						
Min (increment) max	✓						
Value/Phase	✓	✓					
Min (increment) max/phasemin, phasemax			✓				
(Value1, Value2)			✓		✓		
Value, value, ..., value						✓	✓

1 L and C values are replaced by SHORT and OPEN, respectively

[ ] indicates optional information

{ } indicates one element of the group must be selected

b, b_1, b_2 = branch number b_f = "from" branch

n = serial number b_t = "to" branch

n_i = initial node

n_f = final node

Figure 2(e) Parameter Value Formats

When the EX command at the end of the circuit description is accepted, ECAPL requests that the user specify the output desired. The program prints each of the possible output types (e.g., NV) to which the user replies with one of the response forms in Table 1.

Table 1 Output Descriptors

FORM	FUNCTION
numbers	specifies node or branch numbers desired e.g. 1 5 7
$n_1 \rightarrow n_2$	specifies all numbers from n to n inclusive
-n	deletes number n from output list
$-n_1 \rightarrow n_2$	deletes all numbers from n to n inclusive from the output list
0	deletes all previous numbers prior to 0, e.g., 1 3 7 0 2 4 specifies numbers 2 and 4 only.
↑	continuation mark to allow numbers to be continued on next line.
carriage	no output desired

For example, a response to the output type EV might be 1 5 9⟶15 -11 would request element voltages 1, 5, 9, 10, 12, 13, 14, and 15.

The use of ECAPL is best illustrated with a series of examples. These examples demonstrate the interactive procedure and the simplicity of the input language. The first example is a dc analysis of the two-stage amplifier shown in Figure 3 represented by the equivalent circuit shown in Figure 4.

The user requests an ECAPL analysis by typing the statement ECA followed by a carriage return. The system replies with SPECIFY TYPE OF ANALYSIS to which the user responds either DC, AC, or TR. In this case the response was DC. Next, the circuit description is entered as shown in Figure 5. The command EX at the end of circuit description informs ECAPL that the circuit data is complete. The system responds with the number of nodes and branches used in the circuit and a request for the outputs desired. Each output variable is typed by ECAPL and the user must respond with a carriage return (no output) or a specification as described in Table 1.

At the end of the output requests the system dictates the commands to load the appropriate analysis package. A circuit modification (parameter or topology) can be performed at the end of each analysis by reloading the language module, entering the modifications, and loading the analysis module as shown in Figure 5-9.

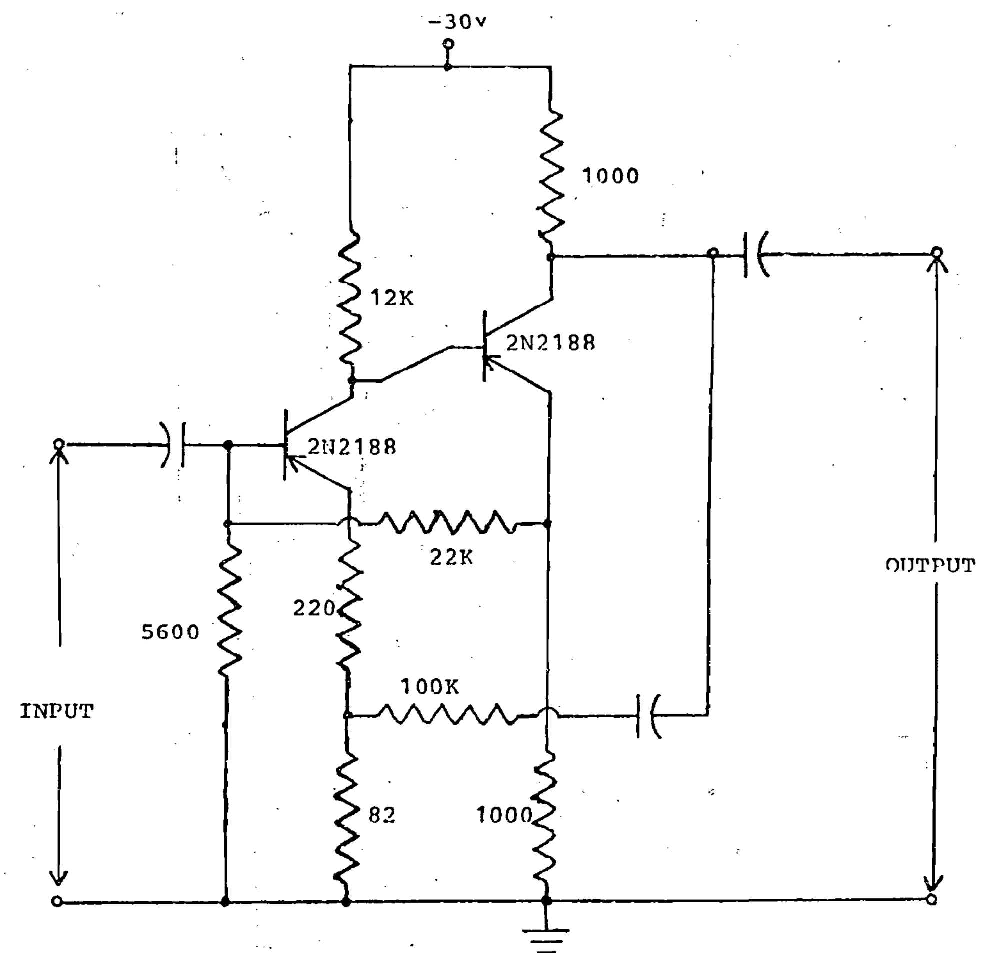


Figure 3 TWO-STAGE TRANSISTOR AMPLIFIER

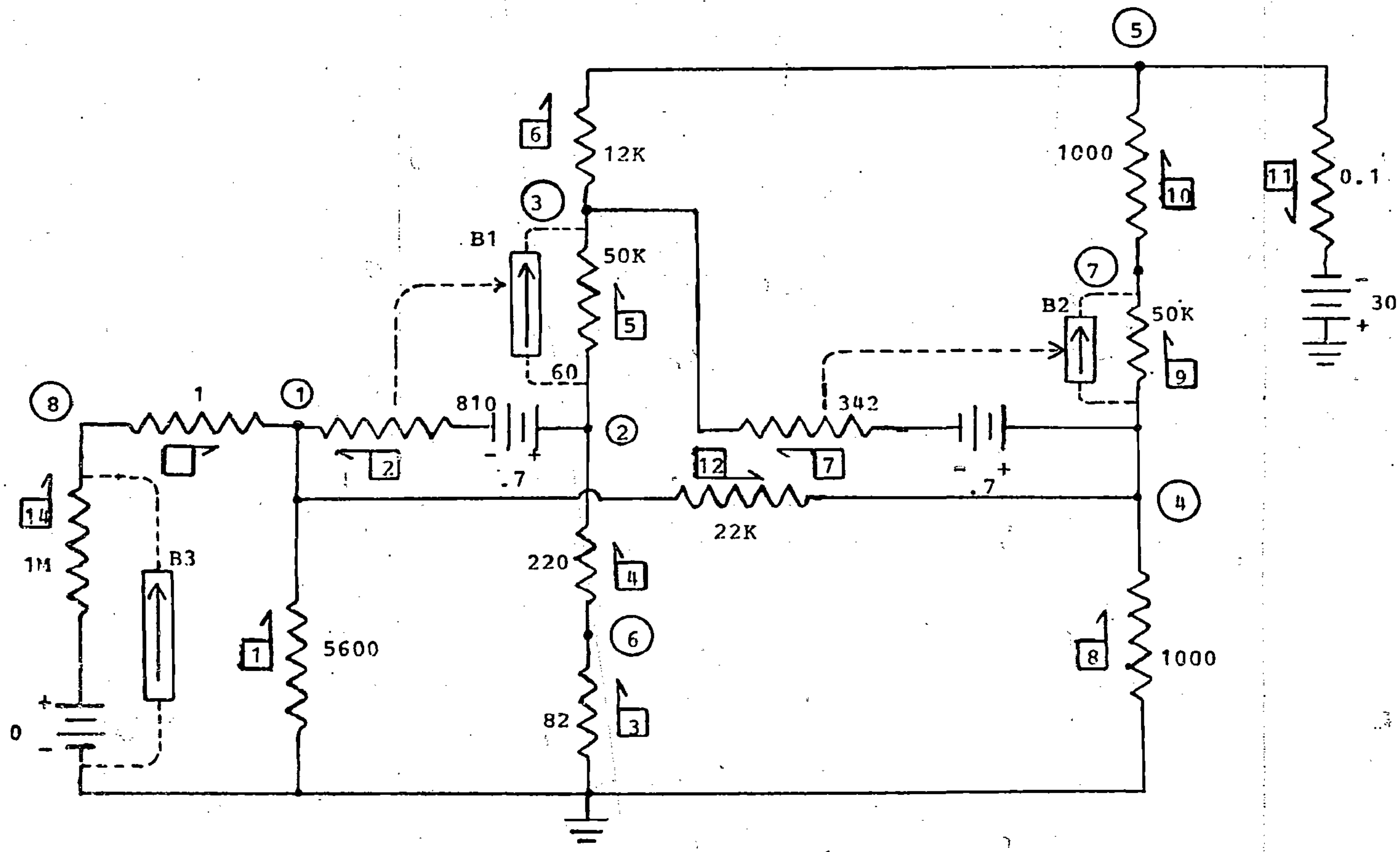


Figure 4 TWO-STAGE AMPLIFIER EQUIVALENT CIRCUIT

Examples of the ac analysis and transient analysis capabilities are illustrated in Figures 6 through 9 respectively.

```
SPECIFY TYPE OF ANALYSIS

DC---TEST PROBLEM (DC ANALYSIS OF TWO STAGE AMPLIFIER)

ENTER CIRCUIT DESCRIPTION

R1,0→1=5.6E3(.05)
R2,2→1=810
E2=¯.7(¯.8,¯.6)
R3,0→6=82(.05)
R4,6→2=220(.05)
R5,2→3=50E3(.05)
R6,3→5=12E3(.05)
R7,4→3=342
E7=¯.7(¯.8,¯.6)
R8,0→4=1E3(.05)
R9,4→7=50E3(.05)
R10,7→5=1E3(.05)
R11,5→0=.1
E11=30(.08)
R12,1→4=22E3(.05)
R13,8→1=1
R14,0→8=1E6
B1,2→5=60
B2,7→9=60
B3,13→14=0
SE
WO
PC
EX

NO. OF BRANCHES: 14
NO. OF NODES:     8


SPECIFY OUTPUT DESIRED

NV
1→8
EV
1→14
EC
1→14
BV
1→14
BC
1→14
EP
1→14
MISC
0

TYPE THE COMMANDS:        )ERASE LANG
                          )COPY TERRYDC DC
                          EX
```

FIGURE 5-1

DC ANALYSIS

....

PARTIAL DERIVITIVES AND SENSITIVITY COEFFICIENTS

WITH RESPECT TO RESISTANCES

NODE	PARTIALS	SENSITIVITY
BRANCH 1		
1	¯2.41432E¯05	¯1.35202E¯03
2	¯2.29332E¯05	¯1.28426E¯03
3	7.36617E¯04	4.12506E¯02
4	¯7.32153E¯04	¯4.10006E¯02
5	¯6.92086E¯08	¯3.87568E¯06
6	¯6.22688E¯06	¯3.48706E¯04
7	¯7.53546E¯04	¯4.21986E¯02
8	¯2.41432E¯05	¯1.35202E¯03
BRANCH 2		
1	¯2.33179E¯[illegible]	¯1.88875E¯04
2	¯3.45[illegible]	¯2.79794E¯05
3	¯[illegible]	¯9.00331E¯[illegible]
4	[illegible]	¯8.949[illegible]
	[illegible]E¯[illegible]	[illegible]
	[illegible].89786E¯	[illegible]
	1.20511E¯	[illegible] 06
[illegible]	1.19885E	[illegible]885E¯06
5	8.84618E	8.84618E¯06
6	5.15314E	5.15314E¯08
7	7.62012[illegible] 03	7.62012E¯06
8	1.97861E¯04	1.97861E¯07
BRANCH 12		
1	6.86271E¯06	1.50980E¯03
2	6.51828E¯06	1.43402E¯03
3	¯2.11257E¯04	¯4.64765E¯02
4	¯2.10042E¯04	¯4.62093E¯02
5	1.87226E¯08	4.11897E¯06
6	1.76986E¯06	3.89370E¯04
7	2.04851E¯04	4.50672E¯02
8	6.86271E¯06	1.50980E¯03

FIGURE 5-4

BRANCH 13

1	¯7.57130E¯10	¯7.57130E¯12
2	¯7.19182E¯10	¯7.19182E¯12
3	2.31003E¯08	2.31003E¯10
4	2.29603E¯08	2.29603E¯10
5	¯2.17038E¯12	¯2.17038E¯14
6	¯1.95275E¯10	¯1.95275E¯12
7	¯2.36312E¯08	¯2.36312E¯10
8	1.26011E¯06	1.26011E¯08

BRANCH 14

1	¯7.57130E¯10	¯7.57130E¯06
2	¯7.19182E¯10	¯7.19182E¯06
3	2.31003E¯08	2.31003E¯04
4	2.29603E¯08	2.29603E¯04
5	¯2.17038E¯12	¯2.17038E¯08
6	¯1.95275E¯10	¯1.95275E¯06
7	¯2.36312E¯08	¯2.36312E¯04
8	¯7.58390E¯10	¯7.58390E¯06

WITH RESPECT TO BETAS

NODE	*PARTIALS*	*SENSITIVITY*

BETA 1

1	1.62414E¯04	9.74485E¯05
2	¯1.84607E¯04	¯1.10764E¯04
3	1.02916E¯02	6.17493E¯03
4	1.02293E¯02	6.13759E¯03
5	¯9.64719E¯07	¯5.78832E¯07
6	¯5.01251E¯05	¯3.00751E¯05
7	¯1.05059E¯02	¯6.30352E¯03
8	1.62414E¯04	9.74484E¯05

BETA 2

1	¯4.99734E¯04	¯2.99840E¯04
2	¯4.79183E¯04	¯2.87510E¯04
3	¯2.42894E¯03	¯1.45737E¯03
4	¯3.03216E¯03	¯1.81929E¯03
5	4.70859E¯07	2.82515E¯07
6	¯1.30109E¯04	¯7.80655E¯05
7	4.91151E¯03	2.94691E¯03
8	¯4.99733E¯04	¯2.99840E¯04

FIGURE 5-5

BETA 3

1	7.57130E¯04	0.00000E00
2	7.19182E¯04	0.00000E00
3	¯2.31003E¯02	0.00000E00
4	¯2.29603E¯02	0.00000E00
5	2.17038E¯06	0.00000E00
6	1.95275E¯04	0.00000E00
7	2.36311E¯02	0.00000E00
8	7.58390E¯04	0.00000E00

WITH RESPECT TO VOLTAGE SOURCES

BRANCH 2

NODE	*PARTIALS*	*SENSITIVITY*
1	8.65085E¯01	6.05560E¯03
2	¯1.28151E¯01	¯8.97057E¯04
3	4.12369E00	2.88558E¯02
4	4.09896E00	2.86927E¯02
5	¯3.82996E¯04	¯2.68097E¯06
6	¯3.47960E¯02	¯2.43572E¯04
7	¯4.17401E00	¯2.92161E¯02
8	8.65084E¯01	6.05559E¯03

BRANCH 7

1	¯2.23968E¯02	¯1.56778E¯04
2	¯2.12302E¯02	¯1.48611E¯04
3	8.56576E¯01	5.99603E¯03
4	¯1.42562E¯01	¯9.97936E¯04
5	2.16883E¯05	1.51818E¯07
6	¯5.76449E¯03	¯4.03514E¯05
7	1.45525E¯01	1.01867E¯03
8	¯2.23968E¯02	¯1.56777E¯04

BRANCH 11

1	¯2.23564E¯02	¯6.70992E¯03
2	¯2.14536E¯02	¯6.43608E¯03
3	¯1.36226E¯01	¯4.08679E¯02
4	¯1.35519E¯01	¯4.06557E¯02
5	¯9.99979E¯01	¯2.99994E¯01
6	¯5.82515E¯03	¯1.74754E¯03
7	¯8.61385E¯01	¯2.58415E¯01
8	¯2.23964E¯02	¯6.70991E¯03

FIGURE 5-6

NODE VOLTAGES

NODE	NOMINAL	MINIMUM	MAXIMUM
1	¯1.26087E00	¯1.47646E00	¯1.06217E00
2	¯5.39041E¯01	¯6.76932E¯01	¯4.17844E¯01
3	¯7.57297E00	¯9.21953E00	¯6.13464E00
4	¯6.83505E00	¯8.40754E00	¯5.47766E00
5	¯2.99991E01	¯3.23992E01	¯2.75990E01
6	¯1.46362E¯01	¯1.85462E¯01	¯1.12352E¯01
7	¯2.30216E01	¯2.67830E01	¯1.90103E01
8	¯1.2609[illegible]	¯1.47645E00	¯1.06217E00

BRANCH VOLTAGES

BRANCH	VALUE
1	1.26087E00
2	7.21833E¯01
3	1.46362E¯01
4	3.92679E¯01
5	7.03393E00
6	2.24261E01
7	7.37927E¯01
8	6.83505E00
9	1.61865E01
10	6.97752E00
11	¯2.99991E01
12	5.57417E00
13	1.26087E¯06
14	1.26087E00

ELEMENT VOLTAGES

BRANCH	VALUE
1	1.26087E00
2	2.18331E¯02
3	1.46362E¯01
4	3.92679E¯01
5	7.03393E00
6	2.24261E01
7	3.79266E¯02
8	6.83505E00
9	1.61865E01
10	6.97752E00
11	8.84637E¯04
12	5.57417E00
13	1.26087E¯06
14	1.26087E00

FIGURE 5-7

ELEMENT CURRENTS

BRANCH	VALUE
1	2.25156E¯04
2	2.69545E¯05
3	1.78490E¯03
4	1.78490E¯03
5	1.75795E¯03
6	1.86885E¯03
7	1.10896E¯04
8	6.83505E¯03
9	6.97752E¯03
10	6.97752E¯03
11	8.84637E¯03
12	2.53371E¯04
13	1.26087E¯06
14	1.26087E¯06

BRANCH CURRENTS

BRANCH	VALUE
1	2.25156E¯04
2	2.69545E¯05
3	1.78490E¯03
4	1.78490E¯03
5	1.75795E¯03
6	1.86885E¯03
7	1.10896E¯04
8	6.83505E¯03
9	6.97752E¯03
10	6.97752E¯03
11	8.84637E¯03
12	2.53371E¯04
13	1.26087E¯06
14	1.26087E¯06

ELEMENT POWERS

BRANCH	VALUE
1	2.83893E¯04
2	5.88502E¯07
3	2.61242E¯04
4	7.00894E¯04
5	1.23553E¯02
6	4.19110E¯02
7	4.20593E¯06
8	4.67178E¯02
9	1.12942E¯01
10	4.86959E¯02
11	7.82582E¯06
12	1.41234E¯03
13	1.58980E¯12
14	1.58980E¯06

FIGURE 5-8

```
SPECIFY TYPE OF ANALYSIS

MODIFY---WORST CASE CALCULATIONS WITH MAX. BETA

ENTER CIRCUIT DESCRIPTION

R2=1044
R7=436
B1=78
B2=78
WO
PC
EX

NO. OF BRANCHES. 14
NO. OF NODES:     8

SPECIFY OUTPUT DESIRED

NV
1→8
EV
1→14
EC
1→14
BV
1→14
BC
1→14
EP
1→14
MISC
0

TYPE THE COMMANDS:      )ERASE LANG
                        )COPY TERRYDC DC
                        EX
```

FIGURE 5-9

DC ANALYSIS (MODIFY)

NODE VOLTAGES

NODE	NOMINAL	MINIMUM	MAXIMUM
1	¯1.26954E00	¯1.48600E00	¯1.06995E00
2	¯5.47419E¯01	¯6.85477E¯01	¯4.26184E¯01
3	¯7.48791E00	¯9.13239E00	¯6.05397E00
4	¯6.75108E00	¯8.32043E00	¯5.39911E00
5	¯2.99991E01	¯3.23992E01	¯2.75990E01
6	¯1.48637E¯01	¯1.87758E¯01	¯1.14630E¯01
7	¯2.30834E01	¯2.68601E01	¯1.90465E01
8	¯1.26954E00	¯1.48600E00	¯1.06996E00

BRANCH VOLTAGES

BRANCH	VALUE
1	1.26954E00
2	7.22120E¯01
3	1.48637E¯01
4	3.98782E¯01
5	6.94049E00
6	2.25112E01
7	7.36831E¯01
8	6.75108E00
9	1.63323E01
10	6.91576E00
11	¯2.99991E01
12	5.48154E00
13	1.26954E¯06
14	1.26954E00

ELEMENT VOLTAGES

BRANCH	VALUE
1	1.26954E00
2	2.21201E¯02
3	1.48637E¯01
4	3.98782E¯01
5	6.94049E00
6	2.25112E01
7	3.68315E¯02
8	6.75108E00
9	1.63323E01
10	6.91576E00
11	8.79170E¯04
12	5.48154E00
13	1.26954E¯06
14	1.26954E00

FIGURE 5-10

ELEMENT CURRENTS

BRANCH	VALUE
1	2.2[illegible]E¯04
2	2.11878E¯05
3	1.81265E¯03
4	1.81265E¯03
5	1.79146E¯03
6	1.87593E¯03
7	8.44758E¯05
8	6.75108E¯03
9	6.91576E¯03
10	6.91576E¯03
11	8.79170E¯03
12	2.49161E¯04
13	1.26954E¯06
14	1.26954E¯06

BRANCH CURRENTS

BRANCH	VALUE
1	2.26703E¯04
2	2.11878E¯05
3	1.81265E¯03
4	1.81265E¯03
5	1.79146E¯03
6	1.87593E¯03
7	8.44758E¯05
8	6.75108E¯03
9	6.91576E¯03
10	6.91576E¯03
11	8.79170E¯03
12	2.49161E¯04
13	1.26954E¯06
14	1.26954E¯06

ELEMENT POWERS

BRANCH	VALUE
1	2.87809E¯04
2	4.68676E¯07
3	2.69426E¯04
4	7.22851E¯04
5	1.24336E¯02
6	4.22296E¯02
7	3.11137E¯06
8	4.55770E¯02
9	1.12950E¯01
10	4.78278E¯02
11	7.72939E¯05
12	1.36578E¯0[illegible]
13	1.61173E¯12
14	1.61173E¯06

FIGURE 5-11

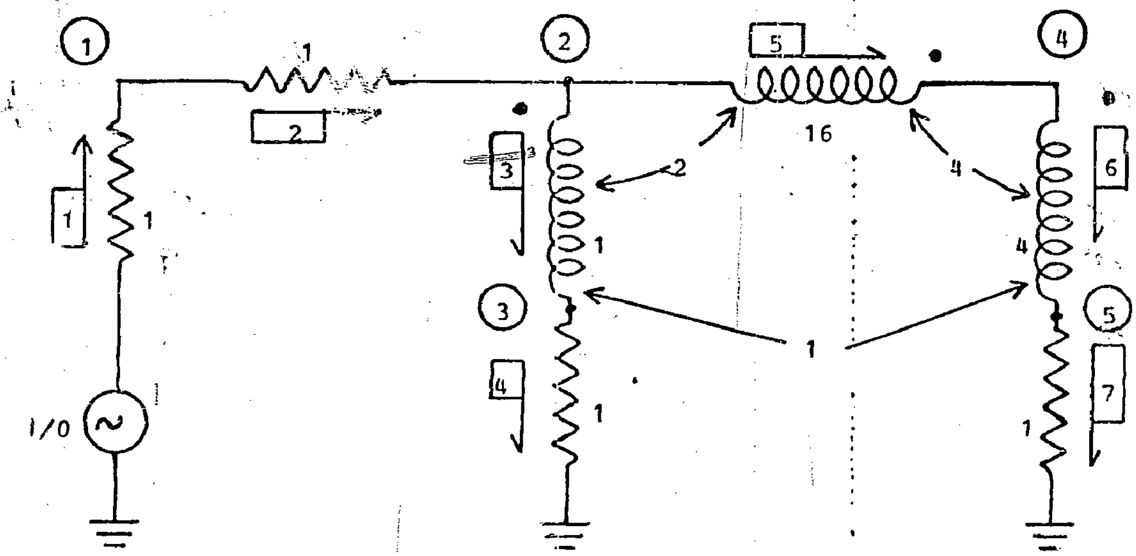


Figure 6 AC ANALYSIS EXAMPLE NETWORK

```
SPECIFY TYPE OF ANALYSIS

AC---TEST PROBLEM

ENTER CIRCUIT DESCRIPTION

R1,0→1=1
E1=1/0
R2,1→2=1
L3,2→3=1
R4,3→0=1
L5,2→4=15
L6,4→5=4
R7,5→0=1
M1,3→6=1
M2,3→5=¯2
M3,5→6=¯4
PC
EX

NO. OF BRANCHES: 7
NO. OF NODES:    5

SPECIFY OUTPUT DESIRED

NV
1→5
EV
1→7
EC
1→7
BV
1→7
BC
1→7
EP
1→7
MISC
1

SPECIFY FREQUENCY (HZ) MIN MAX INCREMENT

.159155

TYPE THE COMMANDS:        )ERASE LANG
                          )COPY ECAPL AC
                          EX
```

FIGURE 7-1

AC ANALYSIS

FREQ= 0.159155

NODE VOLTAGES *NODE*	*MAGNITUDE*	*PHASE*
1	6.70722*E*¯01	8.00
2	3.77764*E*¯01	29.63
3	3.00309*E*¯01	¯12.34
4	2.84899*E*¯01	68.02
5	5.15025*E*¯02	¯34.51

BRANCH VOLTAGES *BRANCH*	*MAGNITUDE*	*PHASE*
1	6.70722*E*¯01	¯172.00
2	3.48548*E*¯01	¯15.54
3	2.53359*E*¯01	82.06
4	3.00309*E*¯01	¯12.34
5	2.34888*E*¯01	¯19.25
6	3.00309*E*¯01	77.66
7	5.15026*E*¯02	¯34.51

ELEMENT VOLTAGES *BRANCH*	*MAGNITUDE*	*PHASE*
1	3.48548*E*¯01	¯15.54
2	3.48548*E*¯01	¯15.54
3	2.53359*E*¯01	82.06
4	3.00309*E*¯01	¯12.34
5	2.34888*E*¯01	¯19.25
6	3.00309*E*¯01	77.66
7	5.15026*E*¯02	¯34.51

ELEMENT CURRENTS *BRANCH*	*MAGNITUDE*	*PHASE*
1	3.48548*E*¯01	¯15.54
2	3.48548*E*¯01	¯15.54
3	3.00309*E*¯01	¯12.34
4	3.00309*E*¯01	¯12.34
5	5.15026*E*¯02	¯34.51
6	5.15026*E*¯02	¯34.51
7	5.15026*E*¯02	¯34.51

FIGURE 7-2

```
BRANCH   CURRENTS
BRANCH        MAGNITUDE            PHASE

  1           3.48548E¯01          ¯15.54
  2           3.48548E¯01          ¯15.54
  3           3.00309E¯01          ¯12.34
  4           3.00309E¯01          ¯12.34
  5           5.15026E¯02          ¯34.51
  6           5.15026E¯02          ¯34.51
  7           5.15026E¯02          ¯34.51

ELEMENT POWERS
BRANCH        MAGNITUDE

  1           1.21485E¯01
  2           1.21485E¯01
  3          ¯5.83554E¯03
  4           9.01857E¯02
  5           1.16711E¯02
  6          ¯5.83554E¯03
  7           2.65252E¯03

YR
  2  ¯1   0   0   0
 ¯1   1   0   0   0
  0   0   1   0   0
  0   0   0   0   0
  0   0   0   0   1


YI
     0            0           0          0          0
     0           ¯1.8437      1.625      0.40625   ¯0.1875
     0            1.625      ¯1.5       ¯0.375      0.25
     0            0.40625    ¯0.375     ¯0.34375    0.3125
     0           ¯0.1875      0.25       0.3125    ¯0.375


ECVR
1  0  0  0  0

ECVI
0  0  0  0  0

LHAT
    1.5          0.125       ¯0.25
    0.125        0.09375      0.0625
   ¯0.25         0.0625       0.375
```

FIGURE 7-3

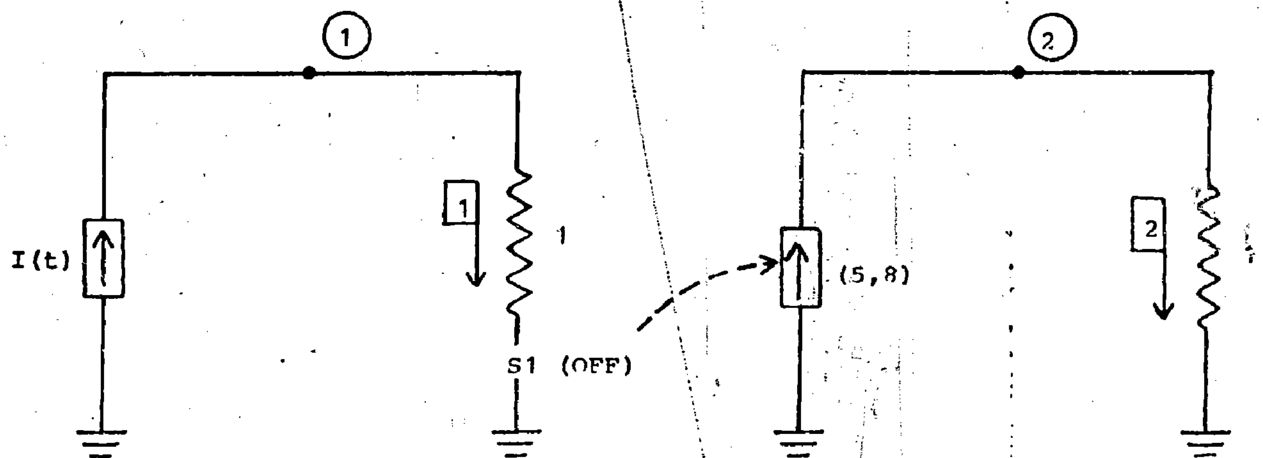


Figure 8 TRANSIENT ANALYSIS EXAMPLE NETWORK

```
SPECIFY TYPE OF ANALYSIS

TRANSIENT

ENTER CIRCUIT DESCRIPTION

R1,0→1=.001
ET1,0,10
SO=10
G2,1→0=1
G3,1→0=(¯1E¯6,¯1.25)
E3=¯1
G4,1→0=(1E¯6,2.25)
E4=¯1.5
SW1,3↑3
SW2,4↓4
TI=1
FI=5
PC
EX

NO. OF BRANCHES: 4
NO. OF NODES:    1


SPECIFY OUTPUT DESIRED

NV
1
EV

EC
1→4
BV

BC

EP

MISC

TYPE THE COMMANDS:       )ERASE LANG
                         )COPY TERRYTRAN TRAN
                         EX
```

FIGURE 9-1

```
TRANSIENT ANALYSIS
..................................

TIME= 0

     NODE VOLTAGES
 NODE          VALUE

   1         4.99500E¯10

     ELEMENT CURRENTS
BRANCH         VALUE

   1        ¯4.99500E¯07
   2         4.99500E¯10
   3         9.99999E¯07
   4        ¯1.50000E¯06

..................................

TIME= 1

     NODE VOLTAGES
 NODE          VALUE

   1         9.99001E¯01

     ELEMENT CURRENTS
BRANCH         VALUE

   1        9.99001E¯01
   2        9.99001E¯01
   3        9.99000E¯10
   4       ¯5.00998E¯07

SWITCHES OFF: 1

..................................

TIME= 1

     NODE VOLTAGES
 NODE          VALUE

   1         9.99976E¯01
```

FIGURE 9-2

```
      ELEMENT CURRENTS
BRANCH          VALUE

   1          1.00001E00
   2          9.99977E¯01
   3          2.93036E¯05
   4         ¯5.00022E¯07

...................................

TIME= 1.4990

      NODE VOLTAGES
 NODE           VALUE

   1            1.49913E00

      ELEMENT CURRENTS
BRANCH          VALUE

   1            8.75219E¯01
   2            1.49912E00
   3           ¯6.23906E¯01
   4           ¯8.74265E¯10

SWITCHES ON: 2
```

FIGURE 9-3

BIBLIOGRAPHY

1. The 1620 Electronic Circuit Analysis Program (ECAP) (1620-EE-02X) User's Manual, White Plains, New York: International Business Machines Corporation, 1965.

USE OF APL IN TEACHING ELECTRICAL NETWORK THEORY

Paul Penfield, Jr.
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

This paper discussed an experiment in which APL was used in a college course about electrical network theory.

Background

The course is 6.01, Introductory Network Theory, which is taught to Freshmen and Sophomores at M.I.T. It covers both continuous networks (RLC networks with dependent sources) and discrete networks, including finite-state machines and combinational-logic networks. In the case of RLC networks, both time-domain and frequency-domain aspects are covered. This course is required for all students majoring in electrical engineering (including computer science) at M.I.T. Except for an elective course in computation, it is the first exposure most of the students have to electrical engineering. It therefore serves as a common foundation for many courses to follow.

There is an enrollment each term of between 100 and 300. Each student is expected to attend two lectures and two recitations per week, and then a one-hour tutorial with a single tutor. In addition, of course, there is homework and in the Spring 1972 term, some of that involved the use of the computer.

Four APL terminals were available sixteen hours per day and this turned out to be sufficient.

Purpose

The APL notation was adopted instead of standard algebraic notation because of its preciseness and completeness. The computer was used partly as motivation for the students, but primarily as a tool to check the correctness of circuit-theory algorithms written by the students.

Note that the purpose is not to introduce the students to a computer, nor to learn a computer language, nor to learn anything about numerical methods or computer-aided design. APL was used simply as a language to express algorithms. Since much of mathematics is, behind the surface, algorithmic, and since much of circuit-theory is mathematical, APL merely served as a language in which to express ideas relating to circuit theory.

Use

APL notation was used in the lectures, in the notes, in the recitations and in the tutorials. Initially, the staff (which consisted of six graduate students and four recitation instructors) was unfamiliar with APL. In the recitations, the use of APL never did take over completely. In the tutorials, whether APL or standard notation was used depended largely on the preferences of the tutors and the individual students. Because the staff was not familiar with APL at first, there were several instances of confusion over precedence rules, or the use of the equals sign. Many of the equations had an unusual appearance. For example, the equation relating the voltage and current in a semiconductor, diode, which in ordinary notation is

$$i = I_s e^{qv/kT}$$

came out in APL notation as

```
I ← IS××Q×V÷K×T
```

which to most of us, appeared relatively awkward at first.

The first lecture, recitation, and homework set exposed the students to the language APL and how to use it on the computer. This exposure turned out to be sufficient. Use was made of *APLCOURSE* in Library 1, and as part of the first homework set, the students were asked to run *APLCOURSE* with a specified set of primitive functions. The students were also given drill

problems to do off-line and asked to verify them at the terminal. The APL User's Manual was a "Suggested Text"; in retrospect, it should have been a required text. In addition, Gilman and Rose, APL 360, An Interactive Approach, was also a "Suggested Text."

After the first week, the students were expected to be able to use the computer when requested. The typical way in which the homework sets were handled is as follows. Consider the algorithms that are necessary to compute the equivalent resistance of two resistors in parallel or two resistors in series. If these two functions, *P* and *S* are dyadic functions with a return, then any series-parallel resistor network can be analyzed by repeated use of these two functions. The students were first asked to solve several series-parallel networks by hand. These were relatively simple networks, consisting of not more than four or five resistors, and element values were chosen so results would usually be integers. Next, the students were asked to write the algorithms for *S* and *P*, in APL, off-line. In the next problem, they were asked to implement these on the computer, and test them by using the examples that they had already solved by hand. Finally, they were asked to find the equivalent resistance of a relatively complicated network with fourteen resistors, which in principle, they could have done by hadn, but in practice, would have been tedious. Other weeks the particular algorithms were different, but the same general approach was used. The students always did simple cases by hand, then wrote the algorithms, then implemented them on the computer, then tested the implementation on the examples that they had already done, and finally, solved a problem that was too complicated to do by hand. About five homework sets out of 13 had such problems.

Results

The use of APL as a notation worked well for the first half of the term. However, when differential equations were encountered, i.e., when RLC networks in the time-domain were introduced, the APL notation was insufficient. The lack of notation for derivatives and integrals proved to be fatal. APL notation was abandoned, although it was returned to at a later time.

Up until this point, however, the APL notation was effective as a communication mechanism, despite the fact that neither the students nor the staff was familiar with it at the outset. Whether it was any more effective than standard notation is unclear; it was certainly no less so, and students had no problems in switching back and forth.

As far as the computer is concerned, the students did the homework sets as they were expected to and they appear to have learned the circuit-theory algorithms by implementing them. The students' experiences are probably best summed up by the answer I received from a number of students when I asked them the question, "Did you find putting the algorithms on the computer to be educational?" Their answer universally was "No" followed by the statement that putting them on the computer did not teach them anything, but writing them prior to putting them on did.

Of course, the use of the computer was not appropriate for all aspects of circuit theory, and was not used every week. In particular, the lack of software dealing with differential equations virtually precluded its use for time-domain analysis of linear networks. However, students did write functions for complex arithmetic, which they then used to implement some frequency-domain analysis techniques for RLC networks.

In general, the students did remarkably well on the computer problems. Of the students who consistently turned in the homework over 90% got the computer problems done correctly most times. Some students did a minimum of work on the computer and some appeared to dislike it, but most did more than was asked and seemed to enjoy the experience and learn from it.

The time spent on the terminals was not excessive. For example, for the series-parallel algorithms discussed above, the typical terminal time was less than one hour per student.

Assessment

My assessment of this experiment, as far as the notation alone is concerned is inconclusive. The lack of notation for derivatives and integrals is a flaw which is very unfortunate because differential equations have an important role in circuit theory.

As far as the student use of the computer is concerned, this was very helpful. The students learned from the compater and they found it enjoyable and provocative.

It should be emphasized that APL was used merely as a vehicle in which to express and test algorithms having to do with circuit theory. Too often it is assumed that the purpose of computers in scientific and engineering courses is to allow the students a "canned" program, since such programs generally do automatically the very steps we want the students to learn. The assumption behind the experiment reported here is that the computer should, instead, be used as

a medium in which the students can express ideas relating to the subject matter. In a sense, the computer then plays the role of a problem grader, forcing a student to continually sharpen his ideas until he "passes," i.e., until his algorithms run properly.

Present plans are to continue the use of APL in future terms, based on the experience reported here.