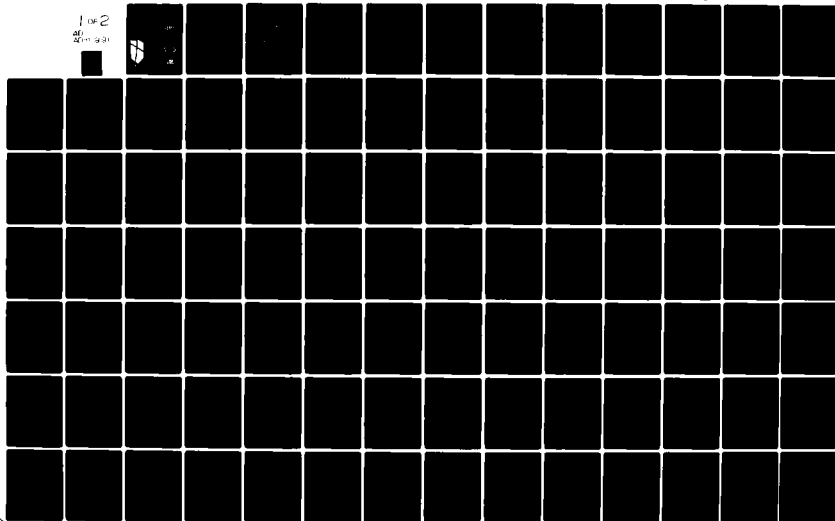
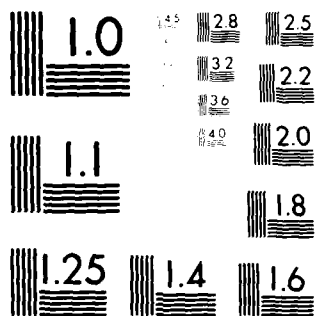


AD-A085 991 CONSTRUCTION ENGINEERING RESEARCH LAB (ARMY) CHAMPAIGN IL F/G 5/1
INTERAGENCY/INTERGOVERNMENTAL COORDINATION FOR ENVIRONMENTAL PL--FTC(U)
MAY 80 R D WFBSTER, D E PUTNAM
UNCLASSIFIED CERL-TR-N-87 NL

1 of 2
of 200-330





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

construction
engineering
research
laboratory



United States Army
Corps of Engineers
... Serving the Army
... Serving the Nation

4

TECHNICAL REPORT N-87
May 1980

INTERAGENCY/INTERGOVERNMENTAL
COORDINATION FOR ENVIRONMENTAL
PLANNING (IICEP): SYSTEMS CONSIDERATIONS

(12) LEVEL II

ADA 085991

by
R.D. Webster
D.E. Putnam

DTIC
ELECTE
JUN 26 1980
S B D

THIS DOCUMENT IS BEST QUALITY PRACTICABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE CORRECTLY.



80 6 26 006

Approved for public release; distribution unlimited.

DDC FILE

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official indorsement or approval of the use of such commercial products. The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

**DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED
DO NOT RETURN IT TO THE ORIGINATOR**

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER 14 CERL-TR-N-87	2. GOVT ACCESSION NO. AD A085991	3. RECIPIENT'S CATALOG NUMBER 9	
4. TITLE (and Subtitle) INTERAGENCY/INTERGOVERNMENTAL COORDINATION FOR ENVIRONMENTAL PLANNING (IICEP): SYSTEMS CONSIDERATIONS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL rept.	
7. AUTHOR(s) R. D. Webster D. E. Putnam		6. PERFORMING ORG. REPORT NUMBER	
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. ARMY CONSTRUCTION ENGINEERING RESEARCH LABORATORY P.O. Box 4005, Champaign, IL 61820		8. CONTRACT OR GRANT NUMBER(s) Project Order No. S-79-26	
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 107		12. REPORT DATE May 1980	
		13. NUMBER OF PAGES 101	
		15. SECURITY CLASS. (of this report) Unclassified	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) 10 Ronald Dwight Webster D.E. Putnam			
18. SUPPLEMENTARY NOTES Copies are obtainable from National Technical Information Service Springfield, VA 22151			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) directories environmental management Air Force state government information systems			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The primary purpose of this report is to document the organization and command structure of a computerized system for providing access to information necessary for the Interagency/Intergovernmental Coordination for Environmental Planning (IICEP) requirements as set forth in Air Force Environmental Planning Bulletin 14. A secondary objective is to identify problems associated with the IICEP system's implementation and to recommend pertinent solutions. Preliminary data acquired			

405279 Jan

Block 20 continued.

by Air Force contractors were obtained and used as a basis for developing the software structure necessary to handle these data. This report describes IICEP and explains the development of the organization, structure, and software of the pilot computerized system. It will form the basis for evaluating the system and further clarifying the need for data base refinement and update.



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

FOREWORD

This project was performed for the Department of the Air Force Engineering and Services Center (AFESC), Tyndall AFB, FL, under Project Order Number S-79-26 dated 19 March 1979. CPT R. Hawkins was the project monitor.

The work was performed by the Environmental Division (EN), U.S. Army Construction Engineering Research Laboratory (CERL), Champaign, IL.

This research was made possible through the efforts of Air Force personnel and the scientists and engineers of CERL. Administrative support and counsel were provided by Dr. E.W. Novak, Acting Chief of EN.

COL L.J. Circeo is Commander and Director of CERL, and Dr. L.R. Shaffer is Technical Director.

ACCESSION for		
NTIS	White Section	☒
DDC	Buff Section	☐
UNANNOUNCED		☐
JUSTIFICATION _____		
BY _____		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	AVAIL. and/or	SPECIAL
A	23 MC	

CONTENTS

	Page
DD FORM 1473	1
FOREWORD	3
1 INTRODUCTION	5
Background	
Objective	
Approach	
2 THE IICEP PROGRAM	5
3 THE PILOT SYSTEM: ORANIZATION AND STRUCTURE	6
4 COMMAND STRUCTURE	8
Selection Commands	
Save and Restore Commands	
List and Peek Commands	
Help and Quit Commands	
5 SUMMARY AND RECOMMENDATIONS	11
REFERENCES	12
APPENDIX A: Sample Data From IICEP Directory	13
APPENDIX B: Software Descriptions	17
APPENDIX C: Source Code	21
DISTRIBUTION	

INTERAGENCY/INTERGOVERNMENTAL COORDINATION FOR ENVIRONMENTAL PLANNING (IICEP): SYSTEMS CONSIDERATIONS

1 INTRODUCTION

Background

The U.S. Army Construction Engineering Research Laboratory (CERL) has maintained an extensive systems development program for Department of Defense (DOD) personnel to use in environmental assessment, planning, and management. These systems include the Environmental Technical Information System (ETIS) and its subsystems¹—the Environmental Impact Computer System (EICS),² the Economic Impact Forecast System (EIFS),³ and the Computer-Aided Environmental Legislative Data system (CELDS).⁴ These systems have been used extensively by both the Army and the Air Force. As a result of this cooperative effort, CERL has been tasked with analyzing new areas for assisting users and producing other systems which respond to these additional requirements and also function in the same interactive mode as ETIS. This mode is extremely beneficial from both developmental and operational standpoints.⁵ For instance, one new application for the ETIS type of system has been the

review and systemization of the Air Force's three-volume directory—Interagency/Intergovernmental Coordination for Environmental Planning (IICEP)—developed to insure adequate coordination of Air Force activities with state and local agencies responsible for environmental planning issues as required by Air Force Interim Planning Bulletin 14. Updating the information in the current directory is a problem. Responsibilities of the listed agencies change constantly; furthermore, the directory—filling three large binders—is physically awkward and inconvenient to update because changes must be mailed to all users. A computerized system could help remedy these difficulties. Implementation of IICEP as a new subsystem of ETIS will encourage maintenance of current directories by simplifying retrieval of the contacts.

Objective

The primary purpose of this research was to develop a pilot IICEP computerized system operating in interactive mode on the same host computer as ETIS and exhibiting the same user-oriented characteristics as the other ETIS subsystems. A secondary objective was to identify any problems associated with the IICEP system's implementation under ETIS and to recommend solutions to these problems.

Approach

The documentation for IICEP was obtained from AFESC, the data base was designed and developed, and an interactive retrieval program was designed and implemented.

2 THE IICEP PROGRAM

IICEP includes a three-volume directory of state environmental planning agencies designed for use by the three Air Force Regional Civil Engineers. Agencies located in all 50 states, Guam, and Puerto Rico are included. The listed agencies deal with issues from the following environmental categories:

- | | |
|----------------------|--------------------|
| 1. General | 7. Noise |
| 2. Air Resources | 8. Socioeconomics |
| 3. Energy | 9. Solid Waste |
| 4. Health and Safety | 10. Transportation |
| 5. Land Use | 11. Water |
| 6. Natural Resources | |

Table 1 gives the subdivisions of the 11 major environmental categories.

¹R.D. Webster, et al., *Development of the Environmental Technical Information System*, Interim Report E-52/ADA009668 (U.S. Army Construction Engineering Research Laboratory [CERL], April 1975).

²Robert Baran and R.D. Webster, *Interactive Environmental Impact Computer System (EICS) User Manual*, Technical Report N-80/ADA074890 (CERL, September 1979).

³R.D. Webster, L. Ortiz, R. Mitchell, and W. Hamilton, *Development of the Economic Impact Forecast System (EIFS)—The Multiplier Aspects*, Technical Report N-35/ADA057936 (CERL, May 1978); J.W. Hamilton and R.D. Webster, *Economic Impact Forecast System, Version 2.0: User's Manual*, Technical Report N-69/ADA073667 (CERL, July 1979).

⁴R.L. Welsh, *User Manual for the Computer-Aided Environmental Legislative Data System*, Technical Report E-78/ADA019018 (CERL, November 1975); J. van Weringh, J. Patzer, R. Welsh, and R. Webster, *Computer-Aided Environmental Legislative Data System (CELDS) User Manual*, Technical Report N-56/ADA061126 (CERL, September 1978).

⁵B.W. Kernighan and J.R. Mashey, "Unix Programming Environment," *Software Practice and Environment*, Vol 9, No. 1 (January 1979), pp 1-15; J. Zucker, K.H. Davis, and P.J. Plauger, *Automated Software Design Tools: "Unix: A High Level Environment for the Development of Microprocessor-Based Systems," "Using Unix for Development of Microprocessor-Based Systems," "Using Unix for Developing Microprocessor Software: A Case Study," "Unix in an Office Environment": presented at Midcon 77 Electronic Show and Convention, Chicago, IL, 8-10 November 1977, Electrical and Electronics Exhibitions, Inc.*

Table 1
Categorical Breakout of IICEP

1. General	6. Natural Resources
Coordination	Land Management and Grounds Maintenance
Environmental Quality	Fish and Wildlife
Environmental Impact Statements	Recreation
A-95 Clearinghouse	Forestry
Transportation	Archaeology and Historic Preservation
2. Air Resources	Flood Control
General	Oil and Gas
3. Energy	7. Noise
General	General
Facility Siting	8. Socioeconomics
4. Health and Safety	Economic Development
General	Education
Civil Defense	Housing
Occupational Health	Local Government
Pesticides	Social Services
Radiation	9. Solid Waste
Building Codes	General
Safety	10. Transportation
5. Land Use	General
Planning	Aeronautics
Agriculture	Highways
Coastal Zone Management	11. Water
Minerals and Geology	General
	Water Resources Management

The IICEP directory contains information allowing the user to decide whether a particular environmental planning issue falls under the responsibility of an agency, and lists the point of contact at each agency. IICEP listings provide the agency's name and function, address, telephone number, and contact person, as shown by the examples in Appendix A. Interim Air Force Environmental Planning Bulletins 14 and 15 have clarified the general concepts of IICEP use and hierarchically organized the data originally contained in the directory. However, CERL's research on IICEP has indicated that updating the information is the most serious problem with the directory and the computerized system, primarily because the jurisdictions and duties of agencies identified in IICEP are vague and change frequently.

Interim Air Force Environmental Planning Bulletin 15, Volume II, lists environmental contacts for Federal agencies. The IICEP pilot program used these contacts as the basis for developing the com-

puter-based retrieval system. The listings contained in the three volumes of contacts for the state environmental planning agencies will be incorporated in the IICEP computer program at a later date.

3 THE PILOT SYSTEM: ORGANIZATION AND STRUCTURE

For the three-volume directory of state environmental planning agencies, Volume II of Interim Air Force Environmental Planning Bulletin 15, and the computerized system, the information in IICEP is organized around a unit of data called a "contact." A given contact consists of information about some individual in the Government, and thus generally lists a name, title, address, and phone number. In addition, a contact includes keyword data which enable an IICEP user to locate the contacts of interest. The keywords currently fall into the following seven categories.

1. "agency"

This category consists of the abbreviated name of the 30 major agencies of the executive branch of the Federal Government. For example, "doc" and "doa" are agency keywords corresponding to the Departments of Commerce and Agriculture. All contacts belonging to the Department of Commerce include "doc" as one of their keywords. A complete list of the agencies and their corresponding keywords is given below:

Advisory Council on Historic Preservation	ach
Civil Aeronautics Board	cab
Community Services Administration	csa
Department of Agriculture	doa
Department of Commerce	doc
Department of Defense	dod
Department of the Interior	doi
Department of Justice	doj
Department of Labor	dol
Department of State	dos
Department of Transportation	dot
Environmental Protection Agency	epa
Energy Research and Development Administration	erd
Executive Office of the President	exo
Farm Credit Administration	fca
Federal Energy Administration	fea
Federal Maritime Commission	fmc
Federal Power Commission	fpc
General Services Administration	gsa
Department of Health, Education and Welfare	hew
Department of Housing and Urban Development	hud
National Aeronautics and Space Administration	nas
Nuclear Regulatory Commission	nrc
National Science Foundation	nsf
National Transportation Safety Board	nts
Small Business Administration	sba
Smithsonian Institution	si
Treasury Department	td
Veterans Administration	va
Water Resources Council	wrc

2. "sub-agency"

This category corresponds to the next level below "agency" in the Government hierarchy. For example, "bureau of the census" is a subagency keyword occurring in some of the contacts belonging to the Department of Commerce.

3. "region"

This category consists of the names of the Federal regions. Unfortunately, many Federal agencies have adopted nonstandard regional divisions. Therefore, these keywords must be taken in the context of the appropriate Federal agency. The 10 standard Federal regions—"region 1" through "region 10"—are included in this category, as well as regions like the "atlanta region" of the Department of Commerce.

4. "state"

This category consists of the 50 state names and the term "us," which refers to the whole United States. Users who seek contacts concerning some issue in an individual state should retrieve the contacts having that state as a keyword (such as Ohio, "oh," or Alabama, "al"), as well as those having "us" as a keyword.

5. "topic"

This category consists of the 11 general topics listed in the following section.

6. "sub-topic"

This is a subcategory of the "topic" category. The following list gives various topics; the subtopics under each are indented.

- a. general
 - coordination
 - environmental quality
 - environmental impact statements/A-95 clearinghouse
 - transportation
- b. air resources
 - general air
- c. energy
 - general energy
 - facility siting
- d. health and safety
 - general health and safety
 - civil defense
 - occupational health
 - pesticides
 - radiation
 - building codes
 - safety

- e. land use
 - planning
 - agricultural
 - coastal zone management
 - minerals and geology
 - f. natural resources
 - land management and ground maintenance
 - fish and wildlife
 - recreation
 - forestry
 - archaeology and historic preservation
 - flood control
 - g. noise
 - general noise
 - h. socioeconomics
 - economic development
 - education
 - housing
 - local government
 - social services
 - i. solid waste
 - general solid
 - j. transportation
 - aeronautics
 - highways
 - k. water
 - general
 - water resources management
7. "program"

This category contains the names of the 26 Air Force programs.

air installation compatible use zone	aicuz
air pollution	ap
airfield and airspace criteria	aac
coastal zone management	czm
compliance with pollution controls	cwpc
comprehensive plan	cp
energy conservation	ec
environmental impact assessments and statements	eias
explosive safety criteria	esc
fish and wildlife and endangered species	fwes
forest management	fm
grazing and agricultural outleasing	gao

interagency/intergovernmental coordination (a-95)	a-95
joint use of military airfields	juma
land management and landscape development	lmlnd
military construction program (programs)	mcpp
military construction program (construction)	mcpc
military family housing	mfh
noise pollution	np
outdoor recreation and cultural resources	orcr
pesticide use and control	puc
real property and acquisition	rpa
real property disposal	rpdp
reducing flight disturbances	rfd
solid waste	sw
water pollution	wp

4 COMMAND STRUCTURE

This chapter discusses the commands available to the IICEP user. Appendices B and C provide the software description and source code for IICEP, respectively, if further clarification is necessary. Table 2 lists and briefly describes IICEP commands.

Selection Commands

The IICEP system maintains a list of all contacts in the data base. By using keywords with the selection commands described below, a user can narrow this list to those contacts of interest.

1. "find"

The "find" command sets the list of contacts equal to those associated with a given keyword. For example, "find doc" sets the current list to contain all the contacts in the Department of Commerce. The find command can be used to retrieve a specific contact. For example, "find # 162" brings the contact number 162 to the current list.

2. "and"

The "and" command limits the current list to contacts already in the list and associated with a given keyword. For example, suppose a user types

"find hew"

"and radiation"

Table 2

Pocket HICEP: Reference For Using HICEP Information Retrieval Program

HICEP Command Glossary

Command Format	Description	Categories (use with "list" and "peek" commands)	Keyword (use with "find," "and," "or" and "except" commands)
find <keyword>	-sets current subset of contacts equal to those associated with given keyword.		
and <keyword>	-limits current subset to those associated with the given keyword.	agency	e.g., epa
or <keyword>	-augments current subset with those associated with the given keyword.	sub-agency	e.g., enforcement
		region	e.g., region 6
except <keyword>	-removes from current subset those contacts associated with the given keyword.	state	e.g., texas
save <filename>	-saves current list in the specified file.	topic	e.g., land use
		sub-topic	e.g., planning
restore <filename>	-replaces current list with list of contacts stored in specified file.	program	e.g., aicuz
restore	-replaces current list with previous list.		
list <category 1> <category 2>	-displays keywords associated with contacts in current list for the given category or categories. (IMPORTANT: if more than one category is to be specified, they should be ordered as follows: < narrower > < broader > e.g., list agency sub-agency)		
peek <category>	-invokes the editor on a copy of system file which contains keywords for given category. "q" returns to HICEP.		
show	-displays contact number, keywords, name, title, address phone number, and possibly comments for each contact in the current list.		
help <term>	-prints message about the given term.		
help	-prints summary of commands and references to more specific topics.		
quit	-exits HICEP program.		

The first command sets the current list to all the contacts in the Department of Health, Education, and Welfare. The second command limits that list to those concerned with radiation. The "and" command can be used to retrieve a specific contact. For example, "find # 234" "and # 678" bring the contact numbers 234 and 678 to the current list.

3. "or"

The "or" command augments the current list to include the contacts associated with a given keyword, and can be used to retrieve a specific contact. For example, suppose a user types

"find us"

"or alabama"

The first command sets the current list to include all contacts having national jurisdiction. The second command expands this list to include contacts with jurisdiction only in Alabama and the Federal government. At this point, the user might wish to further modify the list, for example, by typing

"and radiation"

The three commands create a list of all contacts associated with radiation in Alabama.

4. "except"

The "except" command, which modifies the current list by excluding contacts associated with the given keyword, can be used to exclude specific contacts. For example,

"find radiation"

"except hew"

establishes a list of contacts—other than those in "hew"—associated with radiation. As the selection commands narrow the current list of contacts, the IICEP program reports the number of contacts in the list. When this number is small enough, the user may use the "show" command to see the actual contact data. This command is invoked by simply typing "show" at the keyboard. For each contact in the list, the contact number, keywords, name, title, address, phone number, and any comments are displayed on the terminal. The contact numbers displayed by the "show" command can be used as keywords with any of the selection commands. For example,

"find alabama"

"show"

•
•
•

"except # 435"

"except # 932"

might be used to eliminate contacts that are not of interest.

Save and Restore Commands

Once the user has narrowed contacts down to those of interest, he/she may wish to save this list for future reference. This can be done with the "save" command. For example, if the user types

"save testfile"

the current list is written to a file named "testfile." The file name can be any character string up to 14 characters long. Lists saved in this way can be recovered later by typing

"restore <filename>"

For example,

"restore testfile"

would recover the list saved by the "save testfile" command. The "restore" command can also be used to recover from errors made during the selection process. The "restore" command used without any filename causes the previous list of contacts to be restored as the current list. Suppose a user types

"find texas"

"and alabama"

"restore"

The result of this series of commands is a current list of all contacts associated with "texas" and "alabama." Since these resulted in a null set, the "restore" command reestablished only the set of "texas" contacts without reestablishing the entire search.

List and Peek Commands

The selection commands described earlier are useful only if the user knows which keywords to use. For example, "Bureau of Census" is a keyword, but "Census Bureau" is not. Therefore, two additional commands have been provided to furnish information on keywords. The "list" command displays all the keywords from a given category that apply to the current list of contacts. The number of contacts in the current list which corresponds to a given keyword is displayed alongside that keyword. For example,

"find radiation"

"list agency"

displays all the agencies which have contacts concerned with radiation.

DOC (1)
EPA (5)
ERD (3)
HEW (1)
NRC (21)

Each of these agencies is a "keyword" associated with the subtopic "radiation." If the selection commands have been used to modify the current list of contacts,

"find all"

can be used to set the current list to contain all the contacts in the data base. Thus,

"find all"

"list agency"

produces a list of all agencies in the data base. The "list" command can be invoked with more than one category name. If the user types

"list sub-agency agency"

the program responds with a list of subagencies and the agencies to which they belong. In general, this feature should be used only when the first category is a subcategory of each subsequent category. The program will respond in any case, but the information generated may be misleading. Suppose the user types

"list state agency"

In this case, the first category is not a subcategory of the second, and when the program responds with

"alabama"

"agency: doa"

it only means that "doa" is one of the agencies having a contact where Alabama is a keyword.

The other command designed to provide information on keywords is the "peek" command. This command invokes the operating system's editor on a copy of the IICEP system keyword file for a given

category. As an example,

"peek sub-topic"

invokes the editor on the file of "sub-topic" keywords. Then,

"g/waste/p"

prints a list of all keyword terms which contain the word "waste." Finally,

"q"

quits the editor session and returns the user to the IICEP program.

Help and Quit Commands

A "help" command has been provided to help acclimate the user to using the system. If the user simply types

"help"

the system responds with a message that briefly summarizes the IICEP commands. For further information, the user can type

"help <term>"

and the system will respond with a message providing information about the given term. For example,

"help list"

gives a brief message concerning the use of the list command. Many help messages refer to other terms that can be used with the help command. By following these chains of reference with the "help" command, many questions can be answered without the aid of a manual. The last command that a user must know is the "quit" command. When the user types

"quit"

the IICEP session is ended.

5 SUMMARY AND RECOMMENDATIONS

This report has documented the organization and command structure of a pilot IICEP computerized system operating in interactive mode as a subsystem

of ETIS. The study also identified difficulties in implementing IICEP. The most serious problem with both the IICEP directory and system is the task of updating the information. The jurisdictions and duties of the various identified agencies are nebulous and change constantly.

Nonetheless, the IICEP program could be a valuable source of information to Air Force planners. If the information were maintained in a central data base accessible by remote terminal and capable of supporting interactive usage, the system could be updated constantly with minimal effort, and users could access it from the central source (the interactive system). Incorporating IICEP into ETIS would allow the user to access IICEP's information without having to learn to operate a new system.

It is recommended that selected potential users (the Air Force Regional Civil Engineering offices, for example) use excerpts from this document to analyze the usefulness of the software produced under this research and development effort. The suggestions resulting from such a review could form the basis for modifying and improving the system.

Furthermore, an update procedure could be set up as part of an effort already contemplated for the ETIS operational component now being established for Army users. The additional update of the IICEP data could be integrated into existing procedures for CELDS and EIFS with little increase in long-term operational costs.

REFERENCES

- Baran, Robert, and R.D. Webster, *Interactive Environmental Impact Computer System (EICS) User Manual*, Technical Report N-80/ADA 074890 (U.S. Army Construction Engineering Research Laboratory [CERL], September 1979).
- Hamilton, J.W., and R.D. Webster, *Economic Impact Forecast System, Version 2.0: User's Manual*, Technical Report N-69/ADA073667 (CERL, July 1979).
- Kernighan, B.W., and J.R. Mashey, "UNIX Programming Environment," *Software Practice and Environment*, Vol 9, No. 1 (January 1979), pp 1-15.
- van Wieringen, J., J. Patzer, R. Welsh, and R. Webster, *Computer-Aided Environmental Legislative Data System (CELDS) User Manual*, Technical Report N-56/ADA061126 (CERL, September 1978).
- Webster, R.D., L. Ortiz, R. Mitchell, and W. Hamilton, *Development of the Economic Impact Forecast System (EIFS)—The Multiplier Aspects*, Technical Report N-35/ADA057936 (CERL, May 1978).
- Webster, R.D., et al., *Development of the Environmental Technical Information System*, Interim Report E-52/ADA009668 (CERL, April 1975).
- Welsh, R.L., *User Manual for the Computer-Aided Environmental Legislative Data System*, Technical Report E-78/ADA019018 (CERL, November 1975).
- Zucker, J., K.H. Davis, and P.J. Plauser, *Automated Software Design Tools: "Unix: A High Level Environment for the Development of Microprocessor-Based Systems," "Using Unix for Development of Microprocessor-Based Systems," "Using Unix for Developing Microprocessor Software: A Case Study," "Unix in an Office Environment": presented at Midcon 77 Electronic Show and Convention, Chicago, IL, 8-10 November 1977, Electrical and Electronics Exhibitions, Inc.*

APPENDIX A:

SAMPLE DATA FROM IICEP DIRECTORY

STANDARD FEDERAL REGION V

A. Illinois

1. General

ILLINOIS
General
Environmental Quality

a. Agency

Environmental Protection Agency
2200 Churchill Road
Springfield 62706

(217) 782-3397

Richard H. Briceland, Director

Function—The Agency coordinates programs for air quality, noise, solid waste and water quality.

State Laws—The Agency is established by S.L., Chapter III½, Section 1004.

Federal Laws—(See functional headings.)

ILLINOIS
General
A-95 Clearinghouse

b. Agency

Bureau of the Budget
103 State House
Springfield 62706

(217) 782-4520

Leonard Schaeffer, Director

Function—The Bureau is responsible for reviewing federally financed projects in accordance with A-95 procedures.

State Laws—None identified.

Federal Laws—The Bureau coordinates state review of federally assisted projects pursuant to OMB Circular No. A-95.

ILLINOIS
General
Transportation

c. Agency

Department of Transportation
2300 S. Dirksen Parkway
Springfield 62706

(217) 782-5597

Langhorne Bond, Secretary

Function—The Department plans and develops state transportation systems. It develops and implements mass transit programs, plans airports, promotes transportation safety and constructs and maintains highways.

State Laws—The Department is established by the Civil Administrative Code of 1917.

Federal Laws—(See functional headings.)

2. Air Resources

ILLINOIS
Air Resources
General

a. Agency

Division of Air Pollution Control
Environmental Protection Agency
2200 Churchill Road
Springfield 62706

(217) 782-6514

John Moore, Division Director

Function—The Division administers and enforces state air pollution laws and reviews applications for permits.

State Laws—The Division is established under the Environmental Protection Act of 1970, as amended. The Division operates under the following laws and regulations: Stationary Sources Standards, 1972, as amended; Air Quality Standards, 1973; Episodes Regulations, 1976; Open Burning Regulations, 1971, as amended; and Odors Regulations, 1972.

Function—The Division administers state responsibilities under the Clean Air Act.

ILLINOIS
Air Resources
General

b. Agency

Pollution Control Board
309 West Washington Street
Chicago 60606

(312) 793-3620

Jacob D. Dumelle, Board Chairman

Function—The Board establishes air quality standards and regulations.

State Laws—The Board is established under the Environmental Protection Act of 1970, as amended. The Board operates under the following laws and regulations: General Air Pollution Regulations, 1972, as amended; Stationary Sources Standards, 1972, as amended; Air Quality Standards, 1973; Episodes Regulations, 1976; Open Burning Regulations, 1971, as amended; and Odors Regulations, 1972.

Federal Laws—The Board administers state responsibilities under the Clean Air Act.

3. Energy

ILLINOIS
Energy
General

a. Agency

Division of Energy
Department of Business and Economic Development
222 South College Avenue
Springfield 62702

(217) 782-5784

Sidney M. Marder, Director

Function—The Division conducts energy conservation programs and coordinates energy research within the state. The Division administers fuel allocation programs.

State Laws—The Division is organized under 78-1125, S.L. 1974.

Federal Laws—The Division administers energy conservation plans under the Federal Energy Administration Act of 1974.

ILLINOIS
Energy
General

b. Agency

Interstate Oil Compact Commission
(See Interstate Agency Appendix for details.)

APPENDIX B:

SOFTWARE DESCRIPTIONS

Hashing Subsystem

The hashing subsystem provides a means of looking up character strings in files of keywords. If a string is present in these titles, the lookup mechanism returns identifying data, specifying:

1. The number of keyword file in which the string is found
2. The keyword's number within each file
3. The byte offset of the keyword within each file.

The keyword files are specially formatted text files which are named with some fixed prefix such as "key," followed by a numeric string. Generally, it is best to organize keywords into coherent groups according to file numbers associated with each keyword type. For example, in the IICEP system, one category consists of state names, while another consists of agency names, so these categories should have different file numbers. In the files themselves, keywords are marked by a "#" character in column 1, followed by the keyword string, followed by a terminating ":" character. Characters following the ":" character and characters on subsequent lines are not part of the keyword string. This provides space for comments about the keyword. The next "#" character found in column 1 marks the end of the comments and the beginning of a new keyword.

Thus, the "hash" program sets up a hashtable which allows keyword data to be retrieved, but in order to use the "hash" program, another file must be prepared which itself names the keyword files. This file simply lists one keyword file name per line; the following is a current list of IICEP keyword files.

key.0	key.5
key.1	key.6
key.2	key.7
key.3	
key.4	

Then the "hash" program is invoked by

"hash <file list>"

where "<file list>" is the name of the file discussed above. The keyword files in the <file list> file are opened and read in order; each is scanned for keywords, and identifying data on each keyword are written to a temporary file. This temporary file is an array of struct elements defined as follows:

```
struct marker /* word marker structure layout */
int file; /* keyword file number */
int idnum; /* rel word # within file */
long beginbyte; /* byte offset of work in file */
int hashv[3]; /* hash value
```

Next, the "hash" program calls a subroutine named "maketable," which rearranges the contents of the temporary file into a hashtable. The "hashtable" file is also an array of struct elements defined as above; it is about half-empty at this point in the "hash" program, with the empty slots marked by setting the "file" field equal to -1. Slots occupied by struct elements corresponding to keywords from the keyword files are positioned as follows:

1. The total number of slots in the "hashtable" file minus a maximum overflow allowance defines a modulus.
2. The hash value included in a word marker struct is used to define a long integer.
3. The remainder of the long integer divided by the modulus yields a trial position in the hashtable.
4. The marker struct element is inserted into the first empty slot following the trial position.

The reader should consult the "maketable" subroutine source code (Appendix C) to see the actual mechanics of the temporary file of word markers reorganization into the "hashtable" file.

Once the "hashtable" file has been created, the "lookup" subroutine can be called from a "C" program to retrieve identifying data on any character string. The "lookup" routine computes a trial

position in the hashtable just as in steps 1, 2, and 3 above. Then the hashtable is searched until the first empty slot is encountered. The marker struct elements matching the given keyword are passed back to the calling procedure.

Setup Subsystem

The "setup" program scans files of IICEP data on contacts to prepare for retrieval of this data by the IICEP information retrieval program. The "setup" program is invoked by typing

"setup <file list>"

The argument "<file list>" is a file which names the IICEP files containing contact data. These names should be listed in the "<file list>" file, one per line: for example,

pc.0

pc.62

pc.125

pc.181

pc.241

pc.309

pc.377

pc.444

pc.518

pc.595

pc.678

pc.764

pc.841

pc.937

The files of contact data are named "pc.x," where "x" stands for the contact number of the first contact in the file. It is important that the files listed in "pclist" be ordered so that contacts are encountered in strictly increasing order. Each file contains data

on one or more contacts, and each contact has the format given below:

<contact number>

<category number> : <keyword string>

<category number> : <keyword string>

.

.

.

.

<category number> : <keyword string>

&

<text data, including name, title, address, phone, comments>

In the format description above, the <contact number> field is a numeric string giving the number of the particular contact. Contacts are numbered beginning with zero and must be arranged in increasing order. Gaps are permissible, but tend to slow down the retrieval of data.

In the next section, each line gives a keyword string and the category (e.g., "agency," "region") to which it belongs. Presumably, the given string will be found in the keyword file numbered with the given category number. For example, the string for category two will be found in the keyword file for category two.

The latter section must be terminated by a line consisting of a single "&" character.

Succeeding lines contain text data about the contact; the next line containing a "#" character in the first column marks the beginning of a new contact.

The following is an example of data for a specific contact:

61

0: doc

1: office of the secretary

4: socioeconomics

5: economic development

3: us

&

Jerry Jasinowski, Assistant Secretary for Policy
(8-77)

14th Street, N. W.

Washington, DC 20203

(202) 377-2113

The "setup" program opens and reads the data files in the order they are listed in the "plist" file. As "setup" scans the data, messages are printed, if

1. The numbering of the contacts is not consecutive.
2. A keyword is not present in the alleged keyword file.
3. The "&" line ending the keyword section is missing.

Under any of these conditions, the line number in the file is printed along with an appropriate message.

As the "setup" program scans the data files, the keyword data are digested and written to special files that will later be employed by the retrieval program. For each keyword category, a file named "pkey.X" is created. The "X" stands for the number of the corresponding keyword file. Each file lists the id numbers of the keywords pertaining to the contacts in the data files. A "pkey" file can be thought of as an array of integers. If the keyword id numbers for a given contact have no keywords from a given category, or if there is a gap in the contact numbers, then the -1 entry is still present to signify an empty list of keywords.

The "setup" program also creates an index file as it scans the contact data. Each entry in the index file contains the location of a given contact. This

location consists of

1. The number of the "pc" file in which the contact occurs
2. The byte offset of the beginning of the contact
3. The byte offset of the text data for the contact.

If there is a gap in the numbering of the contacts, the missing entries in the index are marked with a -1 in each of the above three fields.

The Retrieval Program

When the "hash" and "setup" programs have been successfully run, the retrieval program "iicep" can be used. A complete description of the retrieval commands can be found in Chapter 4. The following discussion focuses both on the files required by the "iicep" program and on their functions. Five families of data files are used by the "iicep" program:

1. The "key." files containing keywords and comments
2. The "hashtable" of pointers to the "key." files
3. The "pc." files of textual contact data
4. The "pcndx" file indexing the "pc." files
5. The "pkey" files of keyword id numbers.

When the "iicep" program is invoked, a subroutine named "initlist" is called to construct a list of all the contacts in the data base. This is done by reading the "pcndx" file and noting those entries not marked as being empty. Thus, gaps in the sequence of contacts are detected and left out of the list of contact numbers. The list of contact numbers is represented as an array of integer entries and written to a disk file. An entry of -1 marks the end of the list.

The selection commands "find," "and," "or," and "except" modify this list. Each of these commands takes a keyword as an argument. The hashing lookup mechanism converts the keyword string into data specifying the appropriate keyword category and id number within that category. Next, the appropriate "pkey." file is scanned by the "keypcs"

routine to list those contact numbers in which the given keyword appears. Finally, the "bool" subroutine is called to perform the appropriate logical operation on this list of contact numbers and the previous list of contact numbers.

The "list" command scans the current list of contact numbers and the appropriate "pkey." file to determine which keyword id numbers from a given category are associated with the contacts in the current list. The result is a list of keyword id numbers and the number of contacts in which they appeared. Also listed is the number of a specific contact and the location within that contact where the keyword appeared. When this list is completed, the "pcndx" file is used to locate the contacts where the keywords are listed. The "pc." files are then

opened and read in order to retrieve the actual keyword strings so that they can be printed to the user's terminal.

The "show" command runs through the current list of contact numbers and displays the data for each contact. This is done by finding the location of the contact data in the "pcndx" file and then reading the data from the appropriate "pc." file.

The "help" command uses the hashing lookup mechanism to convert a character string into data specifying the category number and byte offset of the string within the given keyword file. The keyword file is then read, and any comments following the keyword string in that file are displayed on the terminal.

APPENDIX C:

SOURCE CODE

Jul 6 14:09 1979 commandefs.i Page 1

```
1  /* this file is included by both iicep.c and select.c  */
2  /* it defines command numbers for switch statements  */
3
4  #define FIND 0
5  #define AND 1
6  #define JR 2
7  #define EXCEPT 3
8  #define SAVE 4
9  #define RESTORE 5
10 #define SHOW 6
11 #define QUIT 7
12 #define PEEK 8
13 #define HELP 9
14 #define LIST 10
```

Jul 6 14:09 1979 keynames.i Page 1

```
1 char *keynames[]
2 {
3     "agency",
4     "sub-agency",
5     "region",
6     "state",
7     "topic",
8     "sub-topic",
9     "program",
10    0,
11 };
```

Jul 6 14:30 1979 params.i Page 1

```
1 #define KEYNAME "/usr/tmp/iicep/data/key."
2 #define HASHTBL "/usr/tmp/iicep/reference/hashtable"
3 #define PCKEY "/usr/tmp/iicep/reference/pckey."
4 #define PCNDX "/usr/tmp/iicep/reference/pcndx"
5 #define PC "/usr/tmp/iicep/data/pc."
6
7 #define NUMTYP5 7
8
9 #define MESSAGES 7
10
11 #define ALL "all"
```

```

1
2 #define BUFRSIZ 256
3
4 struct keybufr
5 {
6     int descrip; /* file descriptor */
7     int *next; /* next empty slot in bufr */
8     int *endcufr; /* marks end of bufr */
9     int bufr[BUFRSIZ]; /* buffer for keyword id numbers */
10 };
11
12
13 struct marker /* word marker structure layout */
14 {
15     int file; /* keyword file number */
16     int idnum; /* rel word # within file */
17     long beginbyte; /* byte offset of word in file */
18     int hashv[3]; /* hash value */
19 };
20
21
22 struct getlhuf /* for buffered input by line */
23 {
24     int fildes; /* file descriptor of the given file */
25     int nleft; /* number of chars left in buffer */
26     char *nextp; /* pointer to next char in buffer */
27     char buff[512]; /* for buffered reads */
28 };
29
30
31
32
33 #define NDXSIZ 256
34
35 struct ndx
36 {
37     int filenum; /* number of file where entry occurs */
38     long keylines; /* byte offset of keyline section */
39     long datalines; /* byte offset of data section */
40 };
41
42
43 struct ndxbufr
44 {
45     int fidndx; /* descriptor of ndx file */
46     struct ndx *nextndx; /* next open slot in buffer */
47     struct ndx *endndx; /* marks end of buffer */
48     struct ndx bufndx[NDXSIZ]; /* buffer for index entries */
49 };
50
51
52 struct keycheck
53 {
54     int keycount; /* number of occurrences */
55     int pcnum; /* id of pc where found */
56     int keynum; /* number of key in pc keylines */

```

```

57  };
58
59
60  struct keymarker
61  {
62      int keytype;           /* category of the keyword */
63      char *keystring;       /* points to the keyword string */
64  };
65
66
67  #define      MAXKEYS      50      /* max keys per single pc */
68  #define      MAXCHARS    1024     /* max chars in all keys per pc */
69
70  struct keydata
71  {
72      int totkeys;           /* number of keys in a pc */
73      struct keymarker keyptr[MAXKEYS]; /* point to all keys for a pc */
74      char keybuf[MAXCHARS]; /* holds keystings for a pc */
75  };

```

```

1  #
2  /*****
3
4  NAME:
5
6      iicep ( main program )
7
8  FUNCTION:
9
10     Implement the commands of the IICEP system.
11
12  ALGORITHM:
13
14     The program begins by performing certain system initialization tasks.
15     In particular, "iam()" is called to select a unique name for the
16     process, and scratch files are created. The scratch files are used
17     for listing the current and previous lists of contacts and they are
18     initialized to list all the contacts in the data base.
19
20     When the above operations are concluded, the program enters the
21     main command loop where the user is prompted to enter a command and
22     control is transferred to the sub-routine appropriate to executing
23     that command.
24
25  CALLS:
26
27      iam()
28      concat()
29      initlist()
30      resp()
31      copy()
32      execute()
33      table()
34      select()
35      save()
36      restore()
37      show()
38      peek()
39      help()
40      keyword()
41      list()
42
43      Also, Unix routines:
44
45      printf()
46      exit()
47      signal()
48      creat()
49      perror()
50      open()
51      setexit()
52      unlink()
53
54  HISTORY:
55
56      written by Dan Putnam - spring 1979.

```

```

57
58 *****/
59
60
61 #include      "structdefs.i" /* defines getlbuf      */
62 #include      "commandefs.i" /* defines commands */
63 #include      "keynames.i"
64 #include      "params.i"     /* needed for KEYNAME only!! -Dan Putnam */
65
66 char *commtbl[] /* command names, must be consistent with commandefs.i */
67 {
68     "find",
69     "und",
70     "cr",
71     "except",
72     "save",
73     "restore",
74     "show",
75     "quit",
76     "peek",
77     "help",
78     "list",
79     0,
80 };
81
82 struct getlbuf bufin; /* for line oriented input */
83 char line[256]; /* used with bufin */
84 int count; /* character count returned from getl */
85
86 int fidscratch[2]; /* file descriptors */
87 int phase 0; /* used to alternate between files */
88
89 char *keyprefix KEYNAME; /* name of the keyword files */
90
91
92
93 main(argc,argv)
94 int argc;
95 char **argv;
96 {
97     int reset(); /* used with setexit to handle breaks */
98     char buffer[256]; /* buffer for user response */
99     char comm[256]; /* buffer for command string */
100     char *src; /* utility pointer used with copy() */
101     char *dst; /* utility pointer used with copy() */
102     char me[10]; /* buffer for my unique name */
103     char scratch[2][15]; /* names of scratch files */
104     int opcode; /* command number */
105     int quitflag; /* loop control: main command loop */
106     int num; /* number of pcs returned from initlist */
107     struct marker *keyword(); /* returns pointer to keyword data */
108
109
110     signal( 2, 1); /* ignore interrupts */
111
112

```

```

113     printf("Welcome to the IICEP information retrieval program\n");
114     printf("For help, type 'help iicep commands'\n");
115
116
117     /****** create scratch files */
118     /****** we will need to read and write on them, so close and reopen */
119
120
121     if(iam(me) < 0)
122     {
123         printf("can't create unique name, aborting\n");
124         exit();
125     }
126
127     concat(me, "0scratch", scratch[0]);
128     concat(me, "1scratch", scratch[1]);
129
130
131     fidscratch[0] = creat(scratch[0], 0666);
132     fidscratch[1] = creat(scratch[1], 0666);
133
134
135     if( fidscratch[0] < 0 || fidscratch[1] < 0 )
136     {
137         perror("pams, creat");
138         exit();
139     }
140
141     close( fidscratch[0] );
142     close( fidscratch[1] );
143
144     fidscratch[0] = open( scratch[0], 2);
145     fidscratch[1] = open( scratch[1], 2);
146
147     if( fidscratch[0] < 0 || fidscratch[1] < 0 )
148     {
149         perror("pams, open");
150         exit();
151     }
152
153
154     /****** initialize scratch files to list all pcs */
155
156     num = initlist( fidscratch[0] );
157
158     printf("%d contacts in current list\n", num);
159     copyfile( fidscratch[0], fidscratch[1] );
160
161
162
163     /****** this is the main command loop */
164
165     for(quitflag = 0; quitflag == 0;)
166     {
167         setexit();
168         signal(2, reset);

```

```

169         printf("\n\nWhat next?\n");
170
171         resp( buffer );           /* get user response */
172
173         src = buffer;
174         dst = buffer;
175         copy( %src, %dst, 0, sizeof( buffer )); /* omit extra blanks */
176
177
178         /****** see if user wants to execute a Unix comand */
179
180         if( *buffer == '!' )
181         {
182             execute( buffer + 1 );
183             continue;           /* go back to top of command loop */
184         }
185
186
187         /****** ccopy characters into command string */
188
189
190         src = buffer;
191         dst = comm;
192         copy( %src, %dst, ' ', sizeof( comm ));
193
194
195         if( (opcode = table( comm, commtbl)) == -1 )
196         {
197             printf("%s' not a command\n", comm);
198             continue;
199         }
200
201         /****** copy() has left src pointing at command argument string */
202
203         switch( opcode )
204         {
205             /****** these commands select the current pc list */
206
207             case FIND:
208             case AND:
209             case OR:
210             case EXCEPT:
211
212                 signal( 2, 1);           /* ignore interrupts here */
213
214                 select( opcode, src);
215                 break;
216
217             case SAVE:
218
219                 signal( 2, 1);           /* ignore interrupts here */
220                 save( src );
221                 break;
222
223
224

```

```

225         case RESTORE:
226             signal( 2, 1);          /* ignore interrupts here */
227             restore( src );
228             break;
229
230         case SHOW:
231
232             /****** don't ignore interrupts here */
233             show(fidscratch[ phase ], src);
234             break;
235
236         case QUIT:
237             quitflag = 1;             /* this will get us out of loop */
238             break;
239
240         case PEEK:
241             peek( src );
242             break;
243
244         case HELP:
245             help(0, keyword( src ) ); /* L = standard output */
246             break;
247
248         case LIST:
249             list( fidscratch[phase], src);
250             break;
251
252         default:
253             printf("%s" is not yet implemented\n", buffer);
254             break;
255     }
256
257     }
258
259     unlink( me );
260     unlink( scratch[0] );
261     unlink( scratch[1] );
262
263 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      eatdata()
7
8  FUNCTION:
9
10     Read the keyword lines for a contact into a "keydata" struct
11     so that they can be more easily referenced.
12
13  CALLING SEQUENCE:
14
15     int pcid
16     int fidpc
17     long offset
18     struct keydata *pcdata
19
20  PARAMETERS:
21
22     pcid          Accession number of the point of contact whose
23                   data is being read.
24
25     fidpc         File descriptor of the contact data file where the
26                   data for the given pc resides.
27
28     offset        Byte offset of the data in the given file.
29
30     pcdata        Points to the structure which gets the key data
31                   to be read from the file.
32
33  RETURNS:
34
35     nothing.
36
37  ALGORITHM:
38
39     The routine seeks into the file and reads the header.
40     If these operations are successful, the routine enters
41     a loop and reads the keyword lines into the "keydata"
42     struct indicated by "pcdata".
43
44  CALLS:
45
46     seekl()
47     getl()
48     copy()
49
50     Also, Unix routines:
51
52     printf()
53
54
55  CALLED BY:
56

```

```

57         snow()
58         list()
59
60 HISTORY:
61
62         written by Dan Putnam - spring 1979.
63
64         *****/
65
66
67
68 #include      "structdefs.i" /* define getlbuf struct      */
69
70
71
72 eatdata( pcid, fidpc, offset, pcddata)
73 int pcid; /* number of source permit      */
74 int fidpc; /* file descriptor of pc data file */
75 long offset; /* byte offset of data for given pc */
76 struct keydata *pcdata; /* gets lines of keyword data */
77 {
78     int keynumber; /* counts number of keys in pc */
79     struct keymarker *markptr; /* points thru keyptrs of pcddata */
80     char *bufptr; /* points thru keybuf of pcddata */
81     char *endptr; /* points off end of keybuf */
82     char *src; /* utility pointer used with copy() */
83     char *dst; /* utility pointer used with copy() */
84     char *end; /* marks end of pcddata buffer */
85     char tag[100]; /* for grabbing tag off of a line */
86     char line[256]; /* gets line lines from getl() */
87     int nbytes; /* returned from getl */
88     struct getlbuf bufr; /* used by getl() */
89
90
91
92     bufr.fildes = fidpc;
93     bufr.nleft = 0;
94
95     if( offset < 0 || seekl( fidpc, offset) < 0 )
96     {
97         printf("can't seek to data on pc %d\n", pcid);
98         return;
99     }
100
101
102
103     if( (nbytes = getl( line, &bufr)) <= 0 )
104     {
105         printf("can't find data on pc %d\n", pcid);
106         return;
107     }
108
109
110
111     keynumber = 0;
112     bufptr = pcddata -> keybuf;

```

```

113 markptr = pdata -> keyptr;
114 while( (nbytes = getl( line, &buf)) > 0 )
115 {
116     line[ nbytes - 1 ] = 0;          /* replace '\n' by null      */
117
118
119     if( line[0] == '&' )
120     {
121         break;                      /* marks end of keywords      */
122
123         src = line;
124         dst = tag;
125         copy( &src, &dst, ':', sizeof( tag ) );
126
127         if( keynumber > MAXKEYS )
128         {
129             printf("MAXKEY limit exceeded\n");
130             break;
131         }
132
133         markptr -> keystring = bufptr;
134         markptr -> keytype = atoi( tag );
135
136
137         if( copy(&src, &bufptr, 0, &(pdata -> keybuf[MAXCHARS]) - bufptr) < 0 )
138         {
139             printf("MAXCHARS limit exceeded\n");
140             break;
141         }
142
143         keynumber++;
144         markptr++;
145     }
146
147     pdata -> totkeys = keynumber;
148 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      getndx()
7
8  FUNCTION:
9
10     Find the location of the data for a given contact, and
11     return a file descriptor for the data file.
12
13  CALLING SEQUENCE:
14
15     int pcid
16     struct ndx *pc_ptr
17     int getndx()
18
19  PARAMETERS:
20
21     pcid          The accession number of the contact of interest.
22
23     pc_ptr        Points to the index struct to be filled in with the
24                    data giving the location of the given contact.
25
26  RETURNS:
27
28     Returns a file descriptor of the contact data file containing
29     the given contact.
30
31  ALGORITHM:
32
33     This routine may be interrupted if the user hits the "rub-out"
34     key. If this happens, the index file won't get closed. To
35     handle this problem, the descriptor is stored in a static variable.
36     The routine begins by examining this variable to see if it is
37     non-zero. If so, the file is closed and the descriptor is set to
38     zero to mark the file as being closed.
39
40     The routine next opens the index file to read the index struct
41     giving the location data for the given contact. Then, the
42     data file containing the given contact is opened and the
43     file descriptor is returned.
44
45  CALLS:
46
47     Various Unix routines.
48
49     close()
50     seek()
51     read()
52     perror()
53
54  CALLED BY:
55
56     show()

```

```

57         list()
58
59     HISTORY:
60
61         Adapted from the "getndx()" routine of the PAMS system - spring 1979.
62
63     *****/
64
65
66     #include      "structdefs.i"          /* defines pc index structure */
67     #include      "params.i"              /* defines PCNDX */
68
69     getndx(pcid, pc_ptr)
70     int pcid;                               /* number of pc that we want */
71     struct ndx *pc_ptr;                     /* index to pc that we want */
72     {
73         char pcfiler[30];                  /* name of pc file */
74         int file;                          /* pc file number where pc is */
75         long offset;                       /* byte offset into a file */
76         int fidpc;                         /* descriptor of pc file */
77         static int fidndx;                 /* descriptor for pc index file */
78
79
80
81         /***** make sure we close old files before using */
82
83         if( fidndx != 0 )
84         {
85             close( fidndx );
86             fidndx = 0;                     /* and mark it as closed */
87         }
88
89
90
91
92         if( (fidndx = open(PCNDX, 0)) < 0 )
93         {
94             perror("getndx can't open pcndx file");
95             return( -1 );
96         }
97
98         offset = pcid;
99         offset *= sizeof( *pc_ptr );
100         if( seekl(fidndx, offset) < 0 )
101         {
102             perror("getndx can't seek into pc index");
103             return( -1 );
104         }
105
106         if( read(fidndx, pc_ptr, sizeof(*pc_ptr)) < sizeof(*pc_ptr) )
107         {
108             printf("can't read pc index file\n");
109             return( -1 );
110         }
111         close( fidndx );
112         fidndx = 0;                         /* mark it as closed */

```

Jul 6 14:04 1979 getndx.c Page 3

```
113
114     file = pc_ptr -> filenum;      /* this is pc file number */
115     concat(PC, locv(0, file), pcfile);
116
117     if( (fidpc = open(pcfile, 0)) < 0 )
118     {
119         perror("getndx can't open pc file");
120     }
121
122
123     return( fidpc );
124
125 }
```

```

1  #
2  /*****
3
4  NAME:
5
6      initlist()
7
8  FUNCTION:
9
10     Initialize a file to list all the contacts in the data base.
11
12  CALLING SEQUENCE:
13
14     int fidpc
15     int initlist()
16
17  PARAMETERS:
18
19     fidpc      file descriptor of the output list of pc accession
20                numbers.
21
22  RETURNS:
23
24     Returns the number of accession numbers in the output list.
25
26  ALGORITHM:
27
28     The routine opens the index file and reads from it in a loop.
29     Empty index structs are marked by having their "filenum" fields
30     set to -1. Whenever a struct is encountered that is not empty,
31     the corresponding accession number is inserted into the output
32     buffer.
33
34  CALLS:
35
36     Unix routines:
37
38     seek()
39     open()
40     perror()
41     exit()
42     read()
43     write()
44     close()
45
46  CALLED BY:
47
48     icedp ( main program )
49     select()
50
51  HISTORY:
52
53     written by Dan Putnam - spring 1979.
54
55  *****/
56

```

```

57
58 #include      "structdefs.i"
59 #include      "params.i"
60
61 #define        PCSIZ      256
62 #define        NDXSIZ     256
63
64 initlist( fidpc )
65 int fidpc;          /* descriptor of output list of pc ids */
66 {
67     int pcbuf[PCSIZ]; /* output buffer for list of pc ids */
68     int *pcptr;       /* points thru pcbuf */
69     int fidndx;       /* descriptor of input index file */
70     struct ndx ndxbuf[NDXSIZ]; /* input buffer for index file */
71     struct ndx *ndxptr; /* points thru ndxbuf */
72     register int n;    /* fast loop counter */
73     int pcid;         /* id of current pc in index list */
74     int countpc;      /* counts number of pc's in index */
75
76
77     seek( fidpc, 0, 0); /* be sure to start at beginning */
78
79     if( (fidndx = open( PCNDX, 0 )) < 0 )
80     {
81         perror("initlist can't open pcndx");
82         exit();
83     }
84
85     pcid = 0;          /* pc ids begin with zero */
86     countpc = 0;       /* no pc's so far */
87     pcptr = pcbuf;
88
89     while( (n = read( fidndx, ndxbuf, sizeof( ndxbuf ) )) > 0 )
90     {
91         n /= sizeof( *ndxptr ); /* n = # of ndx entries */
92
93         ndxptr = ndxbuf;
94
95         do
96         {
97             if( ndxptr++->filenum != -1 )
98             {
99                 /****** got one! */
100
101                 countpc++;
102                 *pcptr++ = pcid;
103
104                 if( pcptr >= &pcbuf[PCSIZ] )
105                 {
106                     if( write(fidpc, pcbuf, sizeof(pcbuf)) < sizeof(pcbuf) )
107                     {
108                         perror("initlist can't write pc id's");
109                         exit();
110                     }
111
112                     pcptr = pcbuf;

```

```

113         )
114     )
115     )
116     pcid++;          /* bump pcid to id of next index entry */
117     )
118     while( --n );    /* count down on number of entries */
119 )
120
121
122
123
124     /****** flush remaining pc id's in pcbuf */
125
126     *pcptr++ = -1;    /* null terminate list */
127
128     n = (pcptr - pcbuf) * 2;
129
130     if( write( fidpc, pcbuf, n) < n )
131     {
132         perror("initlist can't flush pc list");
133         exit();
134     }
135
136     close( fidndx );
137
138     return( countpc );
139 )

```

```

1  #
2  /*****
3
4  NAME:
5
6      keypcs()
7
8  FUNCTION:
9
10     Make a list of the contacts associated with a given keyword.
11
12  CALLING SEQUENCE:
13
14     int type
15     int fidin
16     int fidout
17     int keypcs()
18
19  PARAMETERS:
20
21     type      Id number of the given keyword.
22
23     fidin     File descriptor of the "pckey." file for the category
24               of the given keyword.
25
26     fidout    File descriptor for the output file which will list
27               the accession numbers of the contacts associated
28               with the given keyword.
29
30  RETURNS:
31
32     Returns the number of contacts associated with the given keyword.
33
34  ALGORITHM:
35
36     The routine reads through the "pckey." file given by the "fidin"
37     file descriptor. Each -1 entry in the file bumps the current
38     pc number by one in order to keep track of which pc accession
39     number is current. When an entry matches the "type" argument,
40     the current pc accession number is inserted into the output
41     buffer. The "previous" variable keeps track of the last accession
42     number to be put into the output list, and the routine checks to
43     be sure that no accession number is inserted twice. This step
44     is necessary in case a keyword has been entered twice in the same
45     contact in the data base.
46
47  CALLS:
48
49     Unix routines:
50
51     seek()
52     read()
53     write()
54
55  CALLED BY:
56

```

```

57         select()
58
59 HISTORY:
60
61         written by Dan Putnam - spring 1979.
62
63         *****/
64
65
66 #define      INSIZ      1024      /* size of input buffer */
67
68 #define      OUTSIZ     256      /* size of out buffer  */
69
70 keypcs(type,fidin,fidout)
71 int type;      /* locate pc's with this key      */
72 int fidin;     /* descriptor for pc key file      */
73 int fidout;    /* descriptor for qualified pc file */
74 {
75
76     int count;      /* counts number of qualified pc's */
77     int inbuf[INSIZ]; /* input buffer for pc key file    */
78     int outbuf[OUTSIZ]; /* output buffer for qualified pc's */
79     int *outptr;     /* points to next open slot in outbuf */
80     int pcid;       /* current pc id number            */
81     int previous;   /* id number of last pc put in outbuf */
82     register int n; /* for loop counting thru pc key list */
83     register int *idptr; /* grabs id numbers out of list */
84     register int idkey; /* equals id # of current key in list */
85
86
87     count = 0;
88     pcid = 0;
89     previous = -1;
90     outptr = outbuf;
91
92     seek(fidin, 0, 0);      /* start at beginning */
93     seek(fidout, 0, 0);    /* start at beginning */
94
95     while( (n = read(fidin, inbuf, INSIZ * 2)) > 0 )
96     {
97         n /= 2;      /* n = # of entries in buffer */
98
99         idptr = inbuf;
100
101         do
102         {
103             if( (idkey = *idptr++) == -1 )
104                 pcid++;
105
106         } else
107         {
108             if( idkey == type && pcid > previous )
109             {
110                 /****** not one! */
111
112                 count++;

```

Jul 6 14:04 1979 keypcs.c Page 3

```
113             previous = poid;    /* to avoid repetition      */
114             *outptr++ = poid;
115
116             if(outptr >= &outbuf[OUTSIZE])
117             {
118                 write(fidout,outbuf,(outptr - outbuf) * 2);
119                 outptr = outbuf;
120             }
121         }
122     }
123     }
124     while( --n );
125 }
126 /****** terminate list and write out      */
127
128 *outptr++ = -1;
129
130 write(fidout,outbuf,(outptr - outbuf) * 2);
131 return(count);
132 }
```

```

1  #
2  /*****
3
4  NAME:
5
6      list()
7
8  FUNCTION:
9
10     Implement the "list" command of IICEP.
11
12  CALLING SEQUENCE:
13
14     int fidpclist
15     char *arg
16
17  PARAMETERS:
18
19     fidsplist      file descriptor of the current list of contact
20                    accession numbers.
21
22     arg            Points to string containing keyword category names
23                    that are to be listed.
24
25  RETURNS:
26
27     nothing.
28
29  ALGORITHM:
30
31     The list command can be interrupted by the user by hitting the
32     "rubout" key. This operation can leave opened files. To deal
33     with this problem, file descriptors are stored in static variables.
34     The routine begins by examining these variables to see if they are
35     non-zero. If so, the files are closed and the descriptors are
36     set to zero to mark the files as being closed.
37
38     The next operation that is performed is to parse the argument
39     string given by "arg". The string is broken down into sub-strings
40     delimited by blanks. The "keynames" array is searched to see
41     if these sub-strings are indeed valid keyword category names.
42     If so, the index in the array which matches a sub-string is saved
43     to identify the category.
44
45     The first category named in the argument string drives the operation
46     of the list command.
47
48     The routine loops to pick up the keys from the first argument
49     category that occur in the current list of contacts.
50     This is accomplished through the call to "listcheck()" which
51     drives the loop. This sub-routine fills out the "checklist" array
52     which keeps track of:
53
54         1. The number of contacts in the current list which contain
55            a given keyword.
56

```

```

57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112

```

2. The accession number of one of the contacts that contains a given keyword.

3. The number of keywords from the given category which precede the keyword in the contact named by item 2 above.

Items 2 and 3 provide a way of recovering a keyword in order to print it. Number 2 gives a contact where it occurs and number 3 indicates which of the keys it is. Since the "listcheck()" array has limited length, it covers just a sub-range of the possible keyword id numbers on each loop iteration. On each call, "listcheck()" returns the smallest id number of a keyword occurring in the current list of contacts which has not yet been considered in the "checklist" array. This provides a lower bound for the next iteration.

Once the "checklist" array has been filled out for an iteration, the routine prints out the keyword data for the checked keys. If the "checklist" struct for a key has not been checked, then nothing is printed. Otherwise, the contact data is read and the given keyword string is printed as it appears in the contact data file. If any other categories were named in the argument list, then the keywords from those categories which occur in the contact data are also printed.

CALLS:

```

copy()
table()
concat()
resp()
listcheck()
getnda()
extdata()

```

Also, Unix routines:

```

close()
printf()
lock()
open()
perror()

```

CALLED BY:

```

show()

```

HISTORY:

written by Dan Putnam - spring 1979.

*****/

```

#include "structdefs.i"
#include "params.i"

```

```

113
114 #define          CHECKSIZ          256
115
116
117
118
119
120 list(fidnclist, arg)
121 int fidpclist;          /* file descriptor of current pc list          */
122 char arg[];             /* contains arguments of list command          */
123 {
124     struct keycheck checklist[CHECKSIZ]; /* marks found keys          */
125     struct keycheck *checkptr;           /* points thru checklist          */
126
127     int type; /* number of chosen category          */
128     int argtype; /* type of other arguments          */
129     int keynumber; /* number of key among keys of a pc          */
130     int i; /* counts keys of a given type          */
131     int occurs; /* number of current pc's with this key          */
132     extern char *keynames[]; /* names of keyword categories          */
133     struct keydata pccdata; /* picks up keyword lines for pcs          */
134     struct keymarker *markptr; /* points thru keymarkers in pccdata          */
135     int arglist[20]; /* argument numbers of show          */
136     int invalid; /* flag = 1 if an argument is invalid          */
137     int argnum; /* loop control: counts arguments          */
138     int num; /* id number of an argument          */
139     char reply[256]; /* gets user response to prompt          */
140     int lo; /* low id in range passed to listcheck          */
141     int hi; /* high id in range passed to listcheck          */
142     char *string; /* points to individual arg strings          */
143     char *src; /* utility pointer used with copy()          */
144     char *dst; /* utility pointer used with copy()          */
145     char *key; /* points to keyword string in pccdata          */
146     register int pcid; /* id number of contact in lists          */
147     struct nda pindex; /* offsets of data in pc file          */
148     char pckeyfile[100]; /* for building pckey filename          */
149     static int fidpc; /* file descriptor for pc data file          */
150     static int fidpckey; /* file descriptor for pckey data file          */
151
152
153
154     /****** make sure file descriptors are closed before using again          */
155
156     if( fidpc != 0 )
157     {
158         close( fidpc );
159         fidpc = 0;
160     }
161
162     if( fidpckey != 0 )
163     {
164         close( fidpckey );
165         fidpckey = 0;
166     }
167
168

```

```

169
170
171
172
173      /****** parse argument string      */
174
175      invalid = 0;                          /* assume all arguments ok */
176      argnum = 0;
177
178
179      src = arg;
180      string = src;
181      dst = src;
182      while( copy( &src, &dst, ' ', 100) > 1 )
183      {
184
185          if( (num = table( string, keynames )) < 0 )
186          {
187              printf("%s" is not a valid argument\n", string);
188              invalid = 1;
189          }
190
191          arglist[ argnum++ ] = num;
192          string = src;                      /* save start of string */
193      }
194
195      if ( invalid )
196          return;                          /* try again */
197
198      arglist[ argnum ] = -1;               /* terminate list of argument codes */
199
200
201      if( argnum == 0 )
202      {
203          return;
204      }
205
206      type = arglist[0];
207
208      concat(pckey, locv( 0, type), pckeyfile);
209
210      if( (fdpckey = open(pckeyfile, 0)) < 0 )
211      {
212          perror("list can't open pckey file");
213          return;
214      }
215
216
217
218
219      lo = 0;
220      hi = CHECKSIZ - 1;
221
222
223      do
224      {

```

```

225     lo = listcheck( fidpclist, fidpckey, checklist, lo, hi);
226     hi = lo + CHECKSIZ - 1;
227
228     for( checkptr = checklist; checkptr < &checklist[CHECKSIZ]; checkptr++)
229     {
230
231         if( checkptr -> keycount == 0 )
232             continue;
233
234
235         pcid = checkptr -> pnum;
236         keynumber = checkptr -> keynum;
237         occurs = checkptr -> keycount;
238
239         if( (fidpc = getno( pcid, &pcindex)) < 0 )
240             continue;
241
242
243         eatdata( pcid, fidpc, pcindex.keylines, &pcdata);
244
245         close( fidpc );
246         fidpc = 0;          /* mark it as being closed          */
247
248         i = 0;
249
250         for( markptr = pcdata . keyptr; markptr++)
251         {
252             if( markptr -> keytype == type )
253             {
254                 if( i++ == keynumber )
255                     break;
256             }
257         }
258
259         printf("%s \(\ %d \)\n", markptr -> keystring, occurs);
260
261         for( argnum = 1; ( argtype = arglist[ argnum ] ) != -1; argnum++)
262         {
263             for( keynumber = 0; keynumber < pcdata . totkeys; keynumber++)
264             {
265                 if( pcdata . keyptr[ keynumber ] . keytype == argtype )
266                 {
267                     key = pcdata . keyptr[ keynumber ] . keystring;
268                     printf("    %s: %s\n\n", keynames[argtype], key);
269                     break;
270                 }
271             }
272         }
273     }
274 }
275 }while( lo >= 0 );
276
277 close( fidpckey );
278 fidpckey = 0;          /* mark it as being closed          */
279 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      listcheck()
7
8  FUNCTION:
9
10     Fill in a checklist indicating the presence of keywords
11     in a list of contacts.
12
13  CALLING SEQUENCE:
14
15     int fidpclist
16     int fidpckey
17     struct keycheck *checklist
18     int lo
19     int hi
20
21  PARAMETERS:
22
23     fidpclist      File descriptor for the list of current contact
24                   accession numbers.
25
26     fidpckey      File descriptor for the "pckey." file listing
27                   keyword id numbers of keywords occurring in
28                   contacts.
29
30     max           Maximum number of keywords per contact
31                   from the given category. Effectively gives
32                   the length of the "rows" of the pckey file.
33
34     checklist     The structs in this array give information
35                   about the occurrence of keywords in the current
36                   contact list:
37
38                   1. The number of permits in the current
39                      list which contain a given keyword.
40
41                   2. The accession number of one of the
42                      contacts that contains a given keyword.
43
44                   3. The number of keywords from the given
45                      category which precede the keyword in the
46                      contact named by item 2 above.
47
48     flag          If this flag is set, only want to get keywords whose
49                   high bits are set to denote non-compliance.
50
51     lo            Defines the low end of the range covered by
52                   the checklist array.
53
54     hi            Defines the high end of the range covered by
55                   the checklist array.
56

```

```

57 RETURNS:
58
59 Returns the smallest id number of the keywords occurring in the current
60 list of contacts, but not yet checked in the "checklist" array.
61 Presumably, this value will be used for the "lo" parameter on the
62 next call to this routine. If there is no such smallest id number,
63 the routine returns -1 to signify that all the keywords have been
64 covered.
65
66 ALGORITHM:
67
68 A pass is made through the "checklist" array to initialize it to
69 empty. Then, the input buffer for the current contact list
70 is filled to prepare for the main loop. In the main loop, each
71 iteration considers an accession number of a contact in
72 the current list of contacts. The routine moves through
73 the "pkey." file to locate the "row" corresponding to the given
74 contact number. The keys listed in this row are checked
75 in the "checklist" provided that they fall into the range
76 defined by "lo" and "hi", and they match the "flag" parameter.
77
78 CALLS:
79
80 seekl()
81
82 Also, Unix calls:
83
84 seek()
85 read()
86 perror()
87
88 CALLED BY:
89
90 list()
91
92 HISTORY:
93
94 written by Dan Putnam - spring 1979.
95
96 *****/
97
98
99
100
101 #include "structdefs.h" /* define keycheck struct */
102
103
104 #define PCLISTSIZ 250 /* buf size for current pc list */
105 #define PCKEYSIZ 768 /* buf size for pkey file */
106
107
108 listcheck( fidpclist, fidpkey, checklist, lo, hi)
109 int fidpclist; /* descriptor for current list of pcs */
110 int fidpkey; /* descriptor for pkey file */
111 struct keycheck checklist[]; /* used to keep track of found keys */
112 int lo; /* key id # of base entry in checklist */

```

```

113 int hi; /* key id # of last entry in checklist */
114 {
115     int newlo; /* smallest key id > hi */
116     int pclist[PCLISTSIZ]; /* buffer for current pc list */
117     int *nclistptr; /* points thru pclist */
118     int *endpclist; /* points off end of pc list */
119     int pkeylist[PKEYSIZ]; /* buffer for pkey file */
120     int *rkeyptr; /* points thru pkey entries */
121     int *endpkey; /* marks end of pkey buffer */
122     int pcid; /* id number of pcs in pclist */
123     int keypc; /* id of pc of pkey entries */
124     int keyval; /* keyword value in pkey file */
125     int keynumber; /* number of keyval in list */
126     int nbytes; /* returned from reads */
127     struct keycheck *checkptr; /* runs thru check list */
128     int i; /* loop control: checklist */
129
130
131
132
133     newlo = 0077777; /* largest pcs integer */
134     keypc = 0; /* pc of first keys */
135
136
137
138
139
140     /****** init checklist to none found */
141
142     checkptr = checklist;
143     for( i = lo; i <= hi; i++)
144     {
145         checkptr->pcnum = -1;
146         checkptr++->keycount = 0;
147     }
148
149
150
151
152     seek( fidpclist, 0, 0); /* be sure to start at the beginning */
153     seek( fidpkey, 0, 0); /* be sure to start at the beginning */
154
155
156     /****** fill up pc list buffer to get started */
157
158     if( (nbytes = read(fidpclist, pclist, PCLISTSIZ * 2)) < 0 )
159     {
160         perror("list can't read current pc list");
161         return;
162     }
163
164     pclistptr = pclist;
165     endpclist = pclist + (nbytes / 2); /* pts off end of list */
166
167
168     /****** fill up pkey buffer to get started */

```

```

169
170     if( nbytes = read(fidpckey, pckeylist, PCKEYSIZ * 2)) < 0 )
171     {
172         perror("list can't read pckey file");
173         return;
174     }
175
176     pckeyptr = pckeylist;
177     endpckey = pckeylist + (nbytes / 2);      /* pts off end of list */
178
179
180
181     /****** run thru current pc list to get keys for each one */
182
183     while( (pcid = *pclistptr++) != -1 )      /* null terminated */
184     {
185         /****** first check if we have used up pc buffer */
186
187         if( pclistptr >= endpclist )
188         {
189             /****** refill buffer and reset pclistptr */
190
191             if( (nbytes = read(fidpclist, pclist, PCLISTSIZ * 2)) < 0 )
192             {
193                 perror("list can't read current pc list");
194                 return;
195             }
196
197             pclistptr = pclist;
198             endpclist = pclist + (nbytes / 2); /* pts off end of list */
199         }
200
201
202         /****** next, read up to proper section of pckey file */
203
204         while( keypc < pcid )
205         {
206             if( *pckeyptr++ == -1 )
207                 keypc++;      /* run thru unwanted key ids */
208
209
210             if( pckeyptr >= endpckey )
211             {
212                 if( (nbytes = read( fidpckey, pckeylist, PCKEYSIZ * 2)) < 0 )
213                 {
214                     perror("list can't read pckey file");
215                     return;
216                 }
217
218                 pckeyptr = pckeylist;
219                 endpckey = pckeylist + (nbytes / 2);
220             }
221
222
223         /****** run thru keys for pcid and put in checklist */
224

```

```

225
226
227     for( keynumber = 0; (keyval = *pkeyp++ ) != -1; keynumber++ )
228     {
229         if( pkeyp >= endpkey )
230         {
231             if( (nbytes = read( fdpkey, pkeylist, FKEYSZ * 2)) < 0 )
232             {
233                 perror("list can't read pkey file");
234                 return;
235             }
236
237             pkeyp = pkeylist;
238             endpkey = pkeylist + (nbytes / 2);
239         }
240
241         if( keyval < lo )
242             continue;
243
244         if( keyval > hi )
245         {
246             if( keyval < newlo )
247                 newlo = keyval;
248
249             continue;
250         }
251
252         checkptr = checklist + ( keyval - lo );
253
254         /****** don't bump count if duplicate keyword in contact */
255
256         if( checkptr -> pnum != pcid )
257         {
258             checkptr -> keycount++;
259             checkptr -> pnum = pcid;
260             checkptr -> keynum = keynumber;
261         }
262
263     }
264
265     keypc = pcid + 1;      /* above loop uses up keys for pcid */
266
267     /****** if newlo has its original value return -1 = all done */
268
269     if( newlo == 077777 )
270         return( -1 );
271     else
272         return( newlo );
273
274 }

```

```

1  #
2  #include      "structdefs.i"
3  #include      "commandefs.i"
4  #include      "params.i"
5
6
7
8
9
10
11 select( opcode, term)
12 int opcode;          /* id number of command */
13 char term[];         /* null terminated string, argument of command */
14 {
15     extern int phase; /* for switching between scratch files */
16     extern int fidscratch[2]; /* file descriptors for scratch files */
17     int fidpckey;      /* descriptor for pckey file */
18     char pckeyfile[256]; /* used to build pckey file name */
19     int old;           /* file descriptor for scratch file */
20     int new;           /* file descriptor for scratch file */
21     char buffer[256];  /* buffer for user response */
22     char *src;         /* utility pointer used with copy() */
23     char *dst;         /* utility pointer used with copy() */
24     struct marker *termpr; /* points to struct describing term */
25     int num;           /* number of pcs from bool or keypcs */
26     int filenum;       /* category of keyword */
27     int idnumber;      /* number of keyword in category */
28     int onepc[2];      /* buffer for writing list of one pc */
29
30     /****** if term is null, use current and old lists */
31     if( term[0] == 0 )
32     {
33         if( opcode != FIND )
34         {
35             /****** note that we don't change phase on this one */
36
37             new = fidscratch[ phase ]; /* new pc list will be in phase */
38             old = fidscratch[ 1 - phase ]; /* out of phase */
39
40             num = bool(old,new,new,opcode);
41             printf("%d in current list\n", num);
42         }
43         return;
44     }
45
46     /****** look at term[0] to see if user wants just one pc */
47     if( term[0] == '#' )
48     {
49         onepc[0] = atoi( term + 1 );
50         onepc[1] = -1;
51
52         phase = 1 - phase; /* maps 0 to 1 and maps 1 to 0 */
53         new = fidscratch[ phase ]; /* new pc list will be in phase */

```

```

57         old = fidscratch[ 1 - phase]; /* old will be out of phase */
58
59         seek( new, 0, 0);
60         if( write( new, onepc, 4) < 4 )
61         {
62             perror("select can't write to pc list");
63             exit();
64         }
65
66         printf("pc %d selected\n", onepc[0]);
67
68         if( opcode != FIND )
69         {
70             num = bool(old,new,new,opcode);
71             printf("%d in current list\n", num);
72         }
73         return;
74     }
75 }
76
77
78
79 /****** special case: user wants list of all pc's */
80
81 if( compar( term, ALL) == 0 )
82 {
83     phase = 1 - phase; /* maps 0 to 1 and maps 1 to 0 */
84     new = fidscratch[ phase ]; /* new pc list will be in phase */
85     old = fidscratch[ 1 - phase]; /* old will be out of phase */
86
87     num = initlist( new );
88
89     printf("%d in current list\n", num);
90
91     if( opcode != FIND )
92     {
93         num = bool(old,new,new,opcode);
94         printf("%d in current list\n", num);
95     }
96     return;
97 }
98
99
100
101 /****** here is where we handle ordinary keywords */
102
103
104 if( (termptr = keyword( term )) != 0 )
105 {
106
107     filenum = termptr -> file;
108     idnumber = termptr -> idnum;
109
110     if( filenum >= MESSAGES )
111     {
112         help(0, termptr ); /* print out message for user */

```

```

113         return;
114     }
115
116     /****** reset phase to switch new and old files */
117
118     phase = 1 - phase;          /* maps 0 to 1 and maps 1 to 0 */
119     new = fidscratch[ phase ];  /* new pc list will be in phase */
120     old = fidscratch[ 1 - phase ]; /* old will be out of phase */
121
122
123
124     concat( PCKEY, locv(0, filenum), pckeyfile);
125
126     if( (fidpkey = open(pckeyfile, C)) < 0 )
127     {
128         perror("select, pckey open");
129         return;
130     }
131
132
133     /****** use keypcs to get list of pcs for non-event keyword */
134
135     num = keypcs(idnumber, fidpkey, new);
136
137
138     printf("%d found\n", num);
139
140     close( fidpkey );
141
142
143     if( opcode != FIND )
144     {
145         num = bool(old,new,new,opcode);
146         printf("%d in current list\n", num);
147     }
148     return;
149 }
150
151
152
153

```

```

1  #
2
3  #include      "structdefs.i"
4  #include      "params.i"
5
6  #define      LSTSIZ      256
7
8
9
10
11
12  show(fid)
13  int fid;      /* file descriptor of current pc list      */
14  {
15      struct getlbuf bufinfo;      /* buffer for getl() routine      */
16      char line[256];      /* gets lines from getl()      */
17      int nbytes;      /* char count from getl      */
18      char *key;      /* points to keyword string      */
19      int type;      /* index to categories      */
20      int printflag;      /* flags printing first key of a type      */
21      struct keymarker *markptr;      /* points thru keyptr array of pcddata      */
22      extern char *keynames[];      /* names of keyword categories      */
23      int keynumber;      /* counts filled in keyptr entries      */
24      struct keydata pcddata;      /* picks up keyword lines for pcs      */
25      char *src;      /* utility pointer used with copy()      */
26      char *dst;      /* utility pointer used with copy()      */
27      int pclist[LSTSIZ];      /* buffer for input and output lists      */
28      register int j;      /* fast loop counter      */
29      register int pcid;      /* id number of source permit in lists      */
30      int *pclistptr;      /* points thru list buffer      */
31      int *endlist;      /* marks end of pclist array      */
32      struct ndx pindex;      /* offsets of data in pc file      */
33      long offset;      /* temp copy of pindex offsets      */
34      static int fidpc;      /* file descriptor for pc data file      */
35
36
37
38      /****** make sure fidpc is closed before using again      */
39
40      if( fidpc != 0 )
41      {
42          close( fidpc );
43          fidpc = 0;      /* mark it as being closed      */
44      }
45
46
47
48      seek(fid,C,0);      /* make sure we get whole file      */
49
50      while( (j = read(fid, pclist, LSTSIZ * sizeof(*pclist))) > 0 )
51      {
52          j /= 2;      /* j = number of ints read      */
53
54          pclistptr = pclist;
55
56          while(j-- && (pcid = *pclistptr++) != -1)

```

```

57     {
58
59
60         if( (fidpc = getndx( pcid, &pcindex)) < 0 )
61             continue;
62
63         printf("pc #%d\n", pcid);
64
65
66         readdata( pcid, fidpc, pcindex.keylines, &pcdata);
67
68         for( type = 0; type < NUMTYPES; type++)
69         {
70             printflag = 0;
71
72             markptr = pcdata . keyptr;
73             for( keynumber = 0; keynumber < pcdata . totkeys; keynumber++)
74             {
75                 if( markptr -> keytype == type )
76                 {
77                     if( printflag++ == 0 )
78                         printf(" \n%s:\n", keynames[ type ]);
79
80                     printf("      %s\n", markptr -> keystring);
81                 }
82                 markptr++;
83             }
84         }
85
86         /****** now print text data */
87
88         printf("\n\n");
89
90         offset = pcindex . datalines;
91
92         if( offset < 0 || seekl( fidpc, offset) < 0 )
93         {
94             printf("can't seek to data\n");
95             continue;
96         }
97
98         bufin.fildes = fidpc;
99         bufin.nleft = 0;
100
101         while( (nbytes = getl( line, &bufin)) > 0 )
102         {
103             line[ nbytes ] = 0;
104
105             if( line[0] == '#' )
106                 break;
107
108             printf("%s", line);
109         }
110
111         close( fidpc );
112         fidpc = 0;

```

/* all done with this one */
/* mark it as being closed */

Jul 6 14:05 1979 show.c Page 3

```
113
114         printf("*****\n\n");
115     }
116
117     if(pcid == -1)
118         break;
119
120 }
121 }
```

```

1  #
2  /*****
3
4  NAME:
5
6      lookup()
7
8  FUNCTION:
9
10     Look in the hash table file for the word marker structs corresponding
11     to a given string.
12
13  CALLING SEQUENCE:
14
15     char *word
16     int fileid
17     int checkflag
18     struct marker *findptr
19     int max
20
21  PARAMETERS:
22
23     word           Points to the string to be looked up in the hashtable.
24
25     fileid         The number of the keyword file in which the word
26                     should be located. If this flag is -1, then all the
27                     keyword files are searched.
28
29     checkflag      If this flag is 0, then a struct whose 3 hash values
30                     match those of the given word is assumed to match the
31                     word. If this flag is non-zero, then the keyword
32                     corresponding to such a struct is read from its
33                     keyword file, and compared to the given word.
34
35     findptr        Points to an array of structs which is filled in
36                     by "lookup()" with the structs which match the given
37                     word.
38
39     max            Gives the size of the above array so that "lookup()"
40                     can avoid overwriting that array.
41
42  RETURNS:
43
44     The number of struct elements matching the given word.
45
46     If an error condition is encountered on an "open()", "seek()" or
47     a "read()", then "exit()" is called to terminate the program.
48
49  ALGORITHM:
50
51     On the first call, the hashtable file is opened and the file
52     descriptor is saved in a static variable to save time on subsequent
53     calls. At this time, "fstat()" is called to determine the length
54     of the hashtable file measured in marker structs. The OVERFLOW
55     parameter is subtracted from this length to determine the "modulus".
56     Obviously, this parameter must agree with its counterpart in the

```

```

57      "hash" program.
58
59      To look up the given word in the hashtable, the "hashfn()" routine
60      is called to compute the 3 hash values of the word. The index
61      into the hash table is computed from the hash values and the "modulus".
62      A calculation is performed to determine the number of structs that
63      can be read beginning with the index, that will not cross a 512 byte
64      boundary in the file. This makes the initial read from the hash table
65      about twice as fast as if it crossed the boundary, and the first
66      read almost always encompasses the collision list.
67
68      The structs in the collision list are scrutinized to see if they
69      match the input word and those that do are copied into the array
70      of structs indicated by "findptr". If this array runs out of room,
71      the structs are no longer copied, but the count of matching structs
72      still continues.
73
74      CALLS:
75
76      hashfn()      To compute hash values of the input word.
77
78      seekl()      To perform seeks at long offsets.
79
80      concat()     To concatenate strings. ( borrowed from CELDS )
81
82      getl()       To read keywords from the keyword files.
83
84      copy()       To extract the keywords from the line on which
85                  they are declared.
86
87      compar()     To compare strings. ( borrowed from CELDS )
88
89      Also, the following Unix calls.
90
91      open()
92      fstat()      To get the size of the hash table file.
93      read()
94      exit()
95      perror()
96      printf()
97
98      CALLED BY:
99
100     Various programs that need to look up keywords.
101
102     HISTORY:
103
104     written by Dan Putnam - fall 1978
105
106     This routine is essentially identical to the "lookup()" used in
107     the PAMS system. The "include" files are the only major
108     difference, and this change was needed only to redefine the
109     "KEYNAME" parameter. Adaptations were made, spring 1979, by
110     Dan Putnam.
111
112     *****/

```

```

113
114
115 #include      "structdefs.h"
116 #include      "params.h"
117
118
119
120 #define      HASHBUFSIZ      32      /* fits in one block      */
121 #define      OVERFLOW      100
122
123 lookup(word,fileid,checkflag,findptr,max)
124 char *word;      /* points to word we are looking for      */
125 int fileid;      /* if -1, any file; if >= 0, specific      */
126 int checkflag;   /* if 1, then check characters      */
127 struct marker *findptr; /* for markers of found words      */
128 int max;         /* length of findptr array      */
129 {
130     struct filestruct /* used for getting length of hashtable      */
131     {
132         char jnk[9]; /* don't need this stuff      */
133         char size0; /* high byte of file size      */
134         int size1; /* low word of file size      */
135         char jnk2[24]; /* don't need this stuff either      */
136     } filedata;
137
138     struct /* used to load size0 and size1 into a long      */
139     {
140         char hi_byte; /* corresponds to size0 of filestruct      */
141         char highest; /* high order byte of a long      */
142         int low_word; /* corresponds to size1 of filestruct      */
143     };
144
145     struct /* used to access high and low words of a long      */
146     {
147         int hibits;
148         int lobits;
149     };
150
151     int num; /* for returning number of finds      */
152
153     register struct marker *srcptr; /*points thru hashtable      */
154     register struct marker *dstptr; /* for moving found markers      */
155
156     struct getlbuf buf; /* struct used by getl() routine      */
157     int nbytes; /* char count returned from getl      */
158     char filename[256]; /* for building keyword file name      */
159     char keyline[256]; /* for reading line from keyword file      */
160     char keystring[256]; /* gets keyword string out of keyline      */
161     char *src; /* used with copy routine      */
162     char *dst; /* used with copy routine      */
163     long index; /* index into hashtable      */
164     long boundary; /* 512 byte boundary after index      */
165     register int readbytes; /* bytes in markers up to boundary      */
166     int hashval[3]; /* hash values      */
167
168     struct marker hashbuf[HASHBUFSIZ]; /* buffer for hashtable      */

```

```

169     struct marker *endbuf;           /* end of markers in hashbuf */
170     int evenword;                    /* number of bytes in hashbuf */
171
172     static int fidhash;              /* descriptor of hashtable */
173     static long modulus;             /* modulus for hash algorithm */
174
175
176
177
178     /****** first call initialization */
179
180     if( fidhash == 0 )
181     {
182         if( (fidhash = open( HASHTBL, 0 )) < 0 )
183         {
184             perror("lookup, can't open hashtable");
185             exit();
186         }
187
188
189         /****** get size of hashtable to compute modulus */
190
191         fstat( fidhash, Rfiledata);
192
193         modulus = 0;
194         modulus.hi_byte = filedata.size0;
195         modulus.low_word = filedata.size1; /* size of hashtable */
196         modulus /= sizeof( *hashbuf ); /* number of keyword markers */
197         modulus -= OVERFLOW;
198     }
199
200
201
202     /****** compute hash values of word and look into hashtable */
203
204     hashfn(word, hashval);
205
206
207     index.lobits = hashval[0];
208     index.hibits = hashval[1] & 0C77777;
209
210
211     index = index % modulus;
212     index *= sizeof( *hashbuf );
213
214     /****** compute number of bytes from index to 512 byte boundary */
215
216     readbytes = 512 - ( index.low_word & 0777 );
217
218     readbytes = (readbytes / sizeof( *hashbuf )) * sizeof( *hashbuf );
219
220     if( readbytes > sizeof( hashbuf ) || readbytes == 0 )
221         readbytes = sizeof( hashbuf );
222
223
224

```

```

225     if(seekl(fidnash,index) < 0)
226     {
227         printf("failed on seek into hashtable\n");
228         exit();
229     }
230
231
232     /****** look at hashtable entries until an empty slot is found */
233
234     num = 0; /* none found so far */
235     dstptr = finoptr; /* copy to register pointer for extra speed */
236
237     while( (nbytes = read( fidhash, hashbuf, readbytes)) > 0 )
238     {
239         readbytes = sizeof( hashbuf ); /* next time fill buffer */
240
241         endbuf = hashbuf + (nbytes / sizeof( *hashbuf ) );
242
243         for( srcptr = hashbuf; srcptr < endbuf; srcptr++)
244         {
245             /****** first check to see if empty */
246
247             if( srcptr -> file == -1 )
248             {
249                 return( num ); /* thats all folks */
250             }
251
252
253             if( srcptr -> hashv[0] != hashval[0] )
254                 continue; /* not found */
255
256             if( srcptr -> hashv[1] != hashval[1] )
257                 continue; /* not found */
258
259             if( srcptr -> hashv[2] != hashval[2] )
260                 continue; /* not found */
261
262
263             if( fileid >= 0 && srcptr -> file != fileid )
264                 continue; /* not in the right file */
265
266
267             if( checkflag )
268             {
269                 /****** check strings to be absolutely sure */
270
271                 concat(KEYNAME, locv(0, srcptr -> file), filename);
272
273
274                 if( (bufr.fildevs = open( filename, 0)) < 0 )
275                 {
276                     perror("lookup can't open keyword file");
277                     exit();
278                 }
279
280                 bufr.nleft = 0;

```

```

281
282         if( seekl( bufr.fildes, srcptr -> beginbyte ) < 0 )
283         {
284             perror("lookup can't seek to keyword");
285             exit();
286         }
287
288
289         if( (nbytes = getl(keyline, &bufr)) < 0 )
290         {
291             printf("lookup can't read keyword file\n");
292             exit();
293         }
294
295         close( bufr.fildes );
296         keyline[nbytes - 1] = 0;
297         src = keyline + 1;
298         dst = keystring;
299         copy( &src, &dst, ':', 256);
300
301         if( compar( keystring, word) != 0 )
302             continue;          /* no match */
303     }
304
305
306     if( num++ < max )
307     {
308         dstptr -> hashv[0] = srcptr -> hashv[0];
309         dstptr -> hashv[1] = srcptr -> hashv[1];
310         dstptr -> hashv[2] = srcptr -> hashv[2];
311         dstptr -> file = srcptr -> file;
312         dstptr -> idnum = srcptr -> idnum;
313         dstptr -> beginbyte = srcptr -> beginbyte;
314         dstptr++;
315     }
316 }
317
318
319     printf("bad read in lookup, index = %D readbytes = %d\n", index, readbytes);
320 }
321

```

```

1  #
2  /*****
3
4  NAME:
5
6      help()
7
8  FUNCTION:
9
10     Print any lines following the line which declares a keyword in
11     a keyword file.
12
13  CALLING SEQUENCE:
14
15     int fid
16     struct marker *termpr
17
18  PARAMETERS:
19
20     fid          File descriptor for output messages. Set to 1
21                  for output to the user's terminal.
22
23     termpr       Points to a word marker struct identifying a given
24                  keyword.
25
26  RETURNS:
27
28     nothing.
29
30  ALGORITHM:
31
32     The routine examines "termpr" and returns immediately if it is
33     a null pointer. Otherwise, the category number is appended to
34     the keyword file prefix and the keyword file is opened. The offset
35     stored in the marker is used to seek into the keyword file.
36     Note that 1 is added to the offset to skip over the '#' character
37     which marks the keyword. This line is not printed, but subsequent
38     lines are printed until a line beginning with '#' is found or
39     until the end of file.
40
41  CALLS:
42
43     concat()
44     seekl()
45     getl()
46
47     Also, Unix calls:
48
49     open()
50     perror()
51     write()
52     close()
53
54  CALLED BY:
55
56     ticep()

```

```

57         select()
58
59     HISTORY:
60
61         written by Dan Putnam - fall 1978 - for PAMS system.
62         Adapted for use by the ICEP system - spring 1979 - by changing the
63         "include" files to define the "KEYNAME" parameter differently.
64
65     *****/
66
67
68
69
70     #include      "structdefs.i"
71     #include      "params.i"
72
73
74     help(fid, termpr )
75     int fid;
76     struct marker *termpr;
77     {
78         char filename[80];
79         char line[80];
80         struct getlbuf buffer;
81         int nchars;
82         int linecount;
83
84
85         /* descriptor of output file */
86         /* describes keyword */
87
88         /* input line from file */
89         /* used by getl routine */
90         /* number of chars in line */
91         /* number of lines printed */
92
93         if( termpr == 0 )
94         {
95             /****** nothing to print */
96
97             return;
98         }
99
100         concat(KEYNAME, locv(0, termpr->file), filename);
101         if( (buffer.fildes = open(filename, 0)) < 0 )
102         {
103             perror("help, can't open");
104             return;
105         }
106
107         buffer.nleft = 0;
108         if( seekl(buffer.fildes, termpr->beginbyte + 1) < 0 )
109         {
110             perror("help, can't seek to keyword\n");
111             close(buffer.fildes);
112             return;
113         }
114
115         linecount = 0;
116         while( (nchars = getl( line, &buffer )) > 0 && line[0] != '#')
117         {
118             if( linecount++ == 0 )

```

Jul 6 14:04 1979 help.c Page 3

```
113         continue;          /* skip over first line      */
114
115         line[nchars] = 0;      /* insert null after the end-of-line. */
116
117         write(fid, line, nchars);
118     }
119
120     close(buffer.fildes);
121 }
```

```
1  #
2  /*****
3
4  NAME:
5
6      iam()
7
8  FUNCTION:
9
10     Create a unique name which can be concatenated with scratch file
11     names to prevent multiple instances of a program from overwriting
12     each others scratch files.
13
14  CALLING SEQUENCE:
15
16      char *me
17      int iam()
18
19  PARAMETERS:
20
21      me - points to a character buffer of at least 3 characters.
22      This buffer receives the unique name, which consists of a lower
23      case letter, followed by a "#" character and a null character.
24
25  RETURNS:
26
27      positive integer      if name creation was successful.
28
29      negative integer      otherwise.
30
31  ALGORITHM:
32
33      The routine uses the "creat" system call to attempt to create
34      a file named with the string "me". The "creat" fails if a file
35      already exists with this name and does not have write access.
36      If this occurs, then the name is altered and the process continues
37      until a unique name is found or else the lower case pre-fixes have
38      been exhausted. In the latter case, -1 is returned to signify
39      failure in creating the unique name. When the procedure succeeds
40      in creating a unique name, the file opened by iam() is closed
41      before returning. It is not expected that this file will be
42      used for anything except to mark its name as already being in use.
43
44  CALLS:
45
46      creat()      Unix system call to create files.
47
48  CALLED BY:
49
50      usually a main program.
51
52  HISTORY:
53
54      written by Dan Putnam - fall 1978
55
56  *****/
```

```

57
58
59
60 iam(me)
61 char *me;
62 {
63     int i;
64     int fid;
65     me[0] = 'a';
66     me[1] = 'h';
67     me[2] = 0;
68     for(i = 0; i < 26; i++)
69     {
70         if( (fid = creat(me,0444)) < 0)
71         {
72             me[i]++;
73         }
74         else
75         {
76             close(fid);
77             return(fid);
78         }
79     }
80
81     return(fid);
82 }

```

```

1  /*          C O N C A T (Note: Borrowed from CELDS, Thank! )
2  *
3  *   Concatenate two strings into one string.  Concat returns a
4  *   pointer to the end of the resultant string so that successive calls
5  *   to concat may be made easily.
6  *
7  * Arguments:  first      pointer to first string
8  *             second     pointer to second string
9  *             result     pointer to end of resulting string
10 *
11 * Returns:    pointer to end of result string
12 *
13 * Calls:      none
14 */
15
16 char *concat (first, second, result) char *first,
17                                         *second,
18                                         *result;
19 {
20     while (*result++ = *first++);      /* Copy first string to result */
21     --result;                          /* Back up over nul */
22     while (*result++ = *second++);     /* Copy second string to result */
23     --result;                          /* Back up over nul */
24     return (result);
25 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      copyfile()
7
8  FUNCTION:
9
10     Copy the contents of one open file to another.
11
12  CALLING SEQUENCE:
13
14     int fidin
15     int filout
16
17  PARAMETERS:
18
19     fidin      File descriptor of source file opened for reading.
20
21     filout     File descriptor of destination file opened for writing.
22
23  RETURNS:
24
25     nothing.
26
27  ALGORITHM:
28
29     The routine first seeks to the start of both files in case other
30     procedures have used the file descriptors. Then the routine reads
31     from the source file in a loop and writes the same number of bytes
32     to the destination as it read.
33
34  CALLS:
35
36     Unix calls:
37
38     seek()
39     read()
40     write()
41     perror()
42
43  CALLED BY:
44
45     pams ( main program )
46     restore()
47     save()
48
49  HISTORY:
50
51     written by Dan Putnam - fall 1978
52
53  *****/
54
55
56

```

```

57  copyfile( fidin, fidout)
58  int fidin;                /* descriptor of source file */
59  int fidout;               /* descriptor of destination file */
60  {
61      int nbytes;
62      char buffer[512];
63
64      seek( fidin, 0, 0);    /* from beginning */
65      seek( fidout, 0, 0);  /* from beginning */
66      while( (nbytes = read( fidin, buffer, 512)) > 0 )
67      {
68          if( write( fidout, buffer, nbytes) < nbytes )
69          {
70              perror("save write error");
71              close( fidout );
72              return;
73          }
74      }
75
76      if( nbytes < 0 )
77      {
78          perror("copyfile, read error");
79      }
80  }
81

```

```

1  #
2  /*****
3
4  NAME:
5
6      resp()
7
8  FUNCTION:
9
10     Get a line of user response form the terminal.
11
12  CALLING SEQUENCE:
13
14     char *bfr
15     int resp()
16
17  PARAMETERS:
18
19     bfr          Points to buffer for user response.
20
21  RETURNS:
22
23     Returns the number of characters in the response exclusive of '\r',
24     or returns -1 on end-of-file.
25
26  ALGORITHM:
27
28     The routine works with a built in limit of 80 characters per response.
29     Characters are read from the terminal until either 80 are read or
30     an end-of-line or end-of-file is encountered. If the last character
31     is an end-of-line, then it is overwritten with a null.
32
33  CALLS:
34
35     nothing.
36
37  CALLED BY:
38
39     pams ( main program )
40     select()
41     keyword()
42     list()
43     geteff()
44     getsmuns()
45
46  HISTORY:
47
48     written by Dan Putnam - fall 1978
49
50  *****/
51
52
53  #define          MAXCHR  80          /* maximum response length          */
54
55
56  resp(bfr)

```

```

57 char *bfr;          /* character buffer for user response */
58 {
59     register int countdown;
60     register int chr;
61     register char *ptr;
62
63
64     countdown = MAXCHR;
65     ptr = bfr;
66
67     do
68     {
69         *ptr++ = chr = getchar();
70
71
72         if( chr == '\0' )
73             return( -1 );
74
75     }
76     while(--countdown && chr != '\n');
77
78     if( chr != '\n' )
79     {
80         while( getchar() != '\n' ); /* flush input */
81         *ptr = 0;
82     }
83     else
84     {
85         *(--ptr) = 0; /* replace CR by null */
86     }
87
88     return( ptr - bfr );
89
90 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      copy()
7
8  FUNCTION:
9
10     Move characters from one string to another and update pointers
11     to source and destination for subsequent calls.
12
13  CALLING SEQUENCE:
14
15     char **source
16     char **dest
17     char delimiter
18     int maxchars
19
20  PARAMETERS:
21
22     source is the address of a pointer to the source character string.
23     this pointer is updated to point past the last character moved.
24
25     dest is the address of a pointer to the destination string.
26     This pointer is left pointing past the null character terminating
27     the string that was moved.
28
29     delimiter is the character signalling the end of the source string.
30     If this character is not encountered, a null character will halt
31     the transfer of characters.
32
33     maxchars is the size of the destination string. If there are
34     more characters to be moved than maxchars, a -1 is returned
35     and copy does not overwrite the end of the buffer.
36
37  RETURNS:
38
39     -1      if the size limitation given by maxchars can not be met.
40
41     otherwise copy returns the number of characters moved including
42     the null character terminating the destination string.
43
44  ALGORITHM:
45
46     The copy routine skips over leading blank or tab characters.
47     Embedded substrings of blanks or tabs in the source string
48     are condensed to one blank. The transfer of characters stops
49     when the delimiter character or a null character is encountered
50     or when the size limitation given by maxchars is met.
51     The source pointer is never moved past a null character.
52     In this case, subsequent calls to copy move an empty string.
53     If the delimiter is not null and it is encountered before a null,
54     then the source pointer is moved past the delimiter.
55     Thus, successive calls can move substrings separated by the delimiter.
56     The destination string is null terminated and the destination

```

```

57         pointer is left pointing past the null. Thus, repeated calls
58         to copy can move strings into a shared buffer.
59
60     CALLS:
61
62         nothing
63
64     CALLED BY:
65
66         all kinds of procedures that move strings around.
67
68     COMMENTS:
69
70         copy() can be used for several different purposes:
71
72         1. cleaning a string to eliminate extra blanks or tabs.
73
74         2. parsing a line into fields.
75
76         3. counting the number of fields on a line.
77
78     HISTORY:
79
80         written by Dan Putnam - fall 1978
81
82     *****/
83
84
85     copy(source, dest, delimiter, maxchars)
86     char **source;          /* points to a pointer to source string */
87     char **dest;            /* points to pointer to destination */
88     char delimiter;         /* stop copying when this char is found */
89     int maxchars;          /* size of destination */
90     {
91         register char *src;    /* copy of source for speed, esthetics */
92         register char chr;     /* temp for *src to save indirection */
93         register int slack;    /* room left in destination */
94         char *dst;            /* points to destination */
95         int ret;              /* return value */
96
97
98
99         src = *source;
100        dst = *dest;
101        slack = maxchars;      /* available room */
102
103        if(slack <= 0)
104        {
105            return(-1);
106        }
107
108
109        /***** first throw away leading blanks and tabs *****/
110
111        while(*src == ' ' || *src == '\t')
112            src++;

```

```

113
114
115
116      /***** now run through the rest of the string *****/
117
118      do
119      {
120          if( (chr = *src) == 0 )
121              break;
122
123          src++;
124
125          if( chr == delimiter )
126              break;
127
128
129          if( chr == ' ' || chr == '\t' )
130              /* if blank or tab */
131              {
132                  while( (chr = *src) == ' ' || chr == '\t' )
133                      src++;
134
135                  if( chr == 0 )
136                      break;
137
138
139                  if( chr == delimiter )
140                  {
141                      src++;
142                      /* move past delimiter */
143                      break;
144                  }
145
146                  chr = ' ';
147              }
148
149          *dst++ = chr;
150
151      } while( --slack );
152
153
154      if( slack > 0 )
155      {
156          ret = maxchars - slack + 1;
157      }
158      else
159      {
160          /***** looks like we didn't find the end but ran out of room */
161
162          --dst;
163          ret = -1;
164
165          /***** move src past delimiter or up to null byte */
166
167          while( (chr = *src) != delimiter && chr != 0 )
168              src++;

```

Jul 5 15:09 1979 /cerl/pams/source/copy.c Page 4

```
169
170         if(chr != 0)
171             src++;
172
173     }
174
175     *dst++ = 0;                /* leave dst pointing past null byte */
176
177     *source = src;
178     *oest = dst;
179     return( ret );
180 }
```

```

1  /*                      E X E C U T E (Note: Borrowed from CELDS, Thanx! )
2  * execute - send a string to sh to be executed
3  *
4  * execute ( command ) ; char *command ;
5  *
6  * Forks off a process to exec the shell with a one-line
7  * command in the string "command".  Waits for return of
8  * the child process.
9  *
10 * Signals are set up so quits will interrupt the child
11 * process, not the parent.
12 *
13 * Calls: fork, signal, exec, wait
14 * Globals: none
15 * Last modification: 31 mar 77
16 *
17 */
18
19 execute (command) char *command;
20 {
21     register int    child,
22                  signalstatus;
23     int    waitstatus;
24
25     if ((child = fork ()) < 0)                /* Set up the fork */
26         return (-1);
27
28     /* The child does the exec using the argument string */
29     if (child == 0) {
30         signal (2, 0);
31         exec ("/bin/sh", "sh", "-c", command, 0);
32     }
33
34     signalstatus = signal (2, 1);
35     while (wait (&waitstatus) != child);    /* Wait for child */
36     signal (2, signalstatus);
37
38     return (0);
39 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      table()
7
8  FUNCTION:
9
10     To look up a character string in an array of string pointers.
11
12  CALLING SEQUENCE:
13
14     char *string
15     char **ptrarray
16     int table()
17
18  PARAMETERS:
19
20     string - points to a null terminated string of characters.
21
22     ptrarray - points to a null terminated array of character pointers.
23
24  RETURNS:
25
26     -1      if the string is not found in the array of pointers.
27
28     otherwise table() returns the index of the first pointer
29     in the array pointing to an identical string.
30
31  EXAMPLE:
32
33     Define "name" and "nametable" as follows:
34
35     char *name      "jody";
36
37     char *nametable[]
38     {
39         "fred",
40         "jody",
41         "pat",
42     }
43
44     Then the call "table( name, nametable)" returns 1 to indicate
45     that "nametable[1]" points to the same string as "name".
46     However, "table( "joe", nametable )" returns -1, since "joe"
47     is not listed in "nametable".
48
49
50  ALGORITHM:
51
52     The "ptrarray" is searched sequentially, and if a pointer in the
53     array points to a string agreeing with that indicated by the
54     "string" argument, then "table" returns the index of that element
55     in the array. If a null pointer is found in the array, then -1
56     is returned.

```

```

57
58 CALLS:
59
60     compar()      A routine borrowed from CELDS to test whether
61                   string pointers point to identical strings.
62
63 CALLED BY:
64
65     usually routines that need to parse command strings or check
66     for "legal" values of string variables from among those in a
67     small, pre-defined list.
68
69 HISTORY:
70
71     written by Dan Putnam - fall 1978
72
73     *****/
74
75
76 table(string,ptrarray)
77 char *string;          /* pts at null terminated string */
78 char **ptrarray;       /* pts at null term array of char ptrs */
79 {
80     register char **ptrptr; /* copy of ptrarray for speed */
81     register char *ptr;    /* copy of *ptrptr for speed */
82     register int i;        /* fast loop counter */
83
84
85     ptrptr = ptrarray;
86     for(i = 0; (ptr = *ptrptr++); i++)
87     {
88         if(compar(string,ptr) == 0)
89             return(i);
90     }
91
92     return(-1);
93 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      save()
7
8  FUNCTION:
9
10     Save the contents of the current scratch file of id numbers in
11     a file named by the input character string.
12
13  CALLING SEQUENCE:
14
15     char *filename
16
17  PARAMETERS:
18
19     filename      Points to the string naming the output file.
20
21  RETURNS:
22
23     nothing.
24
25  ALGORITHM:
26
27     The routine attempts to create a file named by the "filename"
28     argument. If this attempt fails, the routine prints a message
29     to that effect and returns. If it succeeds, then the "copyfile()"
30     routine is used to copy the contents of the current scratch file
31     to the file which has been created.
32
33  CALLS:
34
35     copyfile()
36
37     Also, Unix calls:
38
39     creat()
40     seek()
41     perror()
42
43  CALLED BY:
44
45     pams ( main program )
46
47  HISTORY:
48
49     written by Dan Putnam - fall 1978
50
51  *****/
52
53
54
55  save( filename )
56  char filename[];          /* string naming output file */

```

```
57 {
58     extern int phase;
59     extern int fidscratch[2];
60     int fidin;
61     int fidout;
62
63     fidin = fidscratch[ phase ];
64     seek( fidin, 0, 0 );
65
66     if( (fidout = creat( filename, 0666)) < 0 )
67     {
68         perror("save can't create file");
69         return;
70     }
71
72     copyfile( fidin, fidout );
73
74
75 }
```

```

1  #
2  /*****
3
4  NAME:
5
6      restore()
7
8  FUNCTION:
9
10     Restore a list of data accession numbers to current status.
11
12  CALLING SEQUENCE:
13
14     char *filename
15
16  PARAMETERS:
17
18     filename      Points to string naming the file of accession numbers.
19                   It filename points to a null string, then the previous
20                   list is restored to current status.
21
22  RETURNS:
23
24     nothing.
25
26  ALGORITHM:
27
28     The "filename" parameter is checked to see if it points at a null
29     string. If so, then the global "phase" variable is reset to switch
30     the scratch files. If the "filename" parameter points at a non-null
31     string, then the routine attempts to open the file. If the open
32     is successful, then "phase" is reset and "copyfile()" is called
33     to copy the contents of the input file into the current file.
34
35  CALLS:
36
37     copyfile()
38
39     Also, Unix calls:
40
41     open()
42     perror()
43
44  CALLED BY:
45
46     pams ( main program )
47
48  HISTORY:
49
50     written by Dan Putnam - fall 1978
51
52  *****/
53
54
55
56  restore( filename )

```

```

57 char filename[];          /* names file to be read in.  */
58 {
59     extern int phase;      /* for switching scratch files */
60     extern int fidscratch[2]; /* scratch file descriptors */
61     int fidin;             /* descriptor for restored file */
62     int fidout;            /* copy of scratch descriptor */
63
64
65     if( *filename == 0 )
66     {
67         /****** phase switch effectively restores old list */
68
69         phase = 1 - phase;
70         return;          /* that's all there is to it */
71     }
72
73
74     if( (fidin = open( filename, 0 )) < 0 )
75     {
76         perror("restore can't open file");
77         return;
78     }
79
80     phase = 1 - phase;    /* switch scratch files */
81
82
83     fidout = fidscratch[ phase ]; /* write to in-phase file */
84
85     copyfile( fidin, fidout );
86 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      peek()
7
8  FUNCTION:
9
10     Invoke the editor on the keyword file corresponding to the
11     category name given by the input argument string.
12
13  CALLING SEQUENCE:
14
15     char *category
16
17  PARAMETERS:
18
19     category      String naming the category that the user wants
20                   to inspect.
21
22  RETURNS:
23
24     nothing.
25
26  ALGORITHM:
27
28     The routine begins by checking to see that "category" matches
29     an entry in the "keynames" array. The index of a matching
30     string in that array is the keyword file number of the corresponding
31     file of keywords. This number is appended to the keyword file
32     prefix and the editor is invoked on this file using "execute()".
33
34  CALLS:
35
36     table()
37     concat()
38     execute()
39
40     Also, Unix routines:
41
42     locv()
43
44  CALLED BY:
45
46     pams ( main program )
47
48  HISTORY:
49
50     written by Dan Putnam - fall 1978
51
52  *****/
53
54
55
56  peek( category )

```

```

57 char *category;          /* name of a keyword category */
58 {
59     char command[80];      /* argument string for execute routine */
60     extern char *keyprefix; /* keyword file name prefix */
61     extern char *keynames[]; /* names of categories */
62     int num;               /* category number */
63
64     if( (num = table(category, keynames)) < 0 )
65     {
66         printf("%s' is not a keyword category\n", category);
67         return;
68     }
69
70
71     concat( "ed ", keyprefix, command);
72     concat( command, locv(0,num), command );
73
74     execute( command );
75 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      keyword()
7
8  FUNCTION:
9
10     Lookup a string and prompt for correct category in case of duplicates.
11
12  CALLING SEQUENCE:
13
14     char *term
15     struct marker *keyword().
16
17  PARAMETERS:
18
19     term           Points to the string to be looked up.
20
21
22  RETURNS:
23
24     keyword()      Points to a marker struct which contains data on the
25                     string which has been looked up.
26
27  ALGORITHM:
28
29     Most of the work is done by the "lookup()" routine; this routine
30     is primarily just a user interface to "lookup()". A call to
31     "lookup()" is performed with the parameters set to find all
32     occurrences of the string in the database and check the spelling
33     character for character. If no instances are found, then a message
34     to that effect is printed and a zero pointer is returned.
35     If more than one instance is found, the user is prompted to
36     name which category he wants. A pointer to the appropriate
37     marker struct is returned.
38
39  CALLS:
40
41     lookup()
42     resp()
43
44  CALLED BY:
45
46     letter()
47     pams ( main program )
48     select()
49
50  HISTORY:
51
52     written by Dan Putnam - fall 1978
53
54  *****/
55
56

```

```

57 #include      "structdefs.i"      /* marker decl */
58 #define      MAXFIND      50      /* found array size */
59
60 keyword(term)
61 char *term;      /* lookup this term */
62 {
63     extern char *keynames[];      /* names of keyword categories */
64     char buffer[80];      /* for getting user response */
65     int index;      /* loop control: found words */
66     int filenum;      /* file number of a found word */
67     static struct marker copylist[MAXFIND];      /* identical copies */
68     int copies;      /* counts number of copies */
69
70
71
72
73     /****** lookup: any category, check strings */
74     copies = lookup(term, -1, 1, copylist, MAXFIND);
75
76
77     /****** if copies > 1, prompt for correct category */
78
79     if( copies <= 0 )
80     {
81         printf("can't find '%s'\n", term);
82         return(0);
83     }
84
85     if( copies == 1 )
86     {
87         index = 0;      /* copylist[0] points to the only find */
88     }
89     else
90     {
91         /****** prompt for the correct category */
92
93         printf("Which category?\n");
94
95         for(;;)
96         {
97             for(index = 0; index < copies; index++)
98             {
99                 filenum = copylist[ index ] . file;
100                 printf("Xd: %s\n", index + 1, keynames[ filenum ]);
101             }
102
103             resp( buffer );
104
105             index = atoi( buffer ) - 1;
106
107             if( index >= 0 && index < copies )
108                 break;      /* a valid response */
109         }
110     }
111 }
112

```

Jul 5 15:09 1979 /cerl/pams/source/keyword.c Page 3

```
113  
114     return( &copylist[ index ] );  
115 }
```

```

1  #
2  /*****
3
4  NAME:
5
6      hashfn()
7
8  FUNCTION:
9
10     Convert a null-terminated character string to a 3-word array
11     of integer hash values.
12
13  CALLING SEQUENCE:
14
15     char *string
16     int *hashout
17
18  PARAMETERS:
19
20     string - points to null-terminated string to be hashed.
21
22     hashout - points to output array of 3 hash values.
23
24  RETURNS:
25
26     nothing.
27
28  ALGORITHM:
29
30     One pass is made through the string for each of the 3 output
31     hash values. On a given pass, hashfn() treats the characters
32     in the input string as 5, 6, or 7 bit strings, respectively.
33     This is accomplished by masking off the appropriate number of
34     high order bits in each character ( i.e. 5, 2, or 1 ).
35     The algorithm effectively treats the input string as a bit
36     string, which it "wraps around" the output hash value integer.
37
38     The routine initializes the hash values to zero and processes
39     the input characters until the null terminator is encountered.
40     As each input character is considered, the low order bits
41     are exclusive-ored into the hash value after being shifted
42     past the bits from the previous character. If this results
43     in losing bits off the end of the integer, the lost bits
44     are exclusive-ored onto the low order bits.
45
46     For example, when the first hash value is computed, the
47     first three characters of a string contribute their low order
48     5 bits to give the low order 15 bits of the integer hash value.
49     The low order 5 bits of the next character are exclusive-ored
50     into the output integer as follows. The low order bit is
51     exclusive-ored onto the remaining high bit of the integer, and
52     the next 4 bits are exclusive-ored onto the first 4 bits of the
53     integer. The fifth character is shifted into place beginning
54     with the fifth bit of the output integer.
55
56  CALLS:

```

```

57
58         nothing.
59
60 CALLED BY:
61
62         hash           The program which creates the hashtable file.
63
64         lookup()       The routine used to look up terms in the keyword
65                        files.
66
67 HISTORY:
68
69         written by Dan Putnam - fall 1978
70
71 COMMENTS:
72
73         The three hash values generated by this routine virtually
74         identify strings uniquely. The three hash values are essentially
75         orthogonal in the sense that if two terms collide under one
76         of the hash functions, there is no increased likelihood that
77         they will collide under either of the other two.
78
79         In a file of about 13,000 english words, no two words were
80         found that collided under both of the first two hash functions.
81         When the third function is also considered, it seems virtually
82         assured that if two terms agree in all three hash values, then
83         the two words are identical. If it is assumed that the bit patterns
84         of the hash values are random, it may be computed that the chances
85         of finding a collision in a collection of 10,000 words is less
86         than one in a million.
87
88
89 *****/
90
91
92
93
94 int maskarray[]
95 {
96     057,
97     077,
98     0177,
99 };
100
101 int nbitarray[]
102 {
103     5,
104     6,
105     7,
106 };
107
108 hashfn(string, hashout)
109     char *string;
110     int *hashout;
111 {
112     register int numbits;
113
114     /* string to be hashed */
115     /* 3 word output array */
116     /* number of bits used in hash */

```

```

113     int mask;                /* masks low numbits */
114     char *cpt;               /* points thru string */
115     register int chr;        /* temp copy of *cpt */
116     register int shift;      /* shift chr by this many bits */
117     int hashv;               /* gets hash value */
118     int i;                   /* loop control: 3 hash values */
119
120
121     for( i = 0; i < 3; i++)
122     {
123         mask = maskarray[i];
124         numbits = nbitarray[i];
125         hashv = 0;
126         shift = 0;
127         cpt = strin;
128         while(chr = *cpt++)
129         {
130
131             chr = chr & mask;          /* remove unwanted bits */
132             hashv = (chr << shift);    /* shift into place */
133             shift = (shift + numbits) & 017; /* += numbits mod 16 */
134
135             if(numbits > shift)        /* if we wrap around word */
136
137                 hashv = (chr >> (numbits - shift));
138         }
139
140         hashout[i] = hashv;
141     }
142 }
143

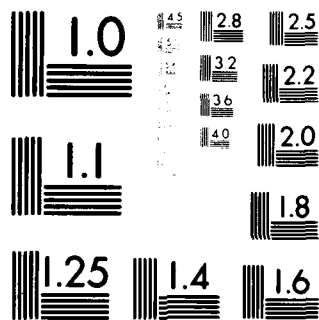
```

AD-A085 991 CONSTRUCTION ENGINEERING RESEARCH LAB (ARMY) CHAMPAIGN IL F/G 5/1
INTERAGENCY/INTERGOVERNMENTAL COORDINATION FOR ENVIRONMENTAL PL--FTC(U)
MAY 80 R D WEBSTER, D E PUTNAM
UNCLASSIFIED CERL-TR-N-87 NL

2 of 2
AD-A085 991



END
DATE
FILMED
8-80
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

```

1  #
2  /*****
3
4  NAME:
5
6      seekl()
7
8  FUNCTION:
9
10     Perform seeks into files with long offsets.
11
12  CALLING SEQUENCE:
13
14      int fid
15      long offset
16
17  PARAMETERS:
18
19      fid is the file descriptor of an open file.
20
21      offset is the offset from the beginning of the file to which
22      seekl() will seek.
23
24  RETURNS:
25
26      returns the same value as the seek() system call returns to seekl().
27      -1 signals an error condition.
28
29  ALGORITHM:
30
31      seekl() tests offset to see if the seek can be performed as an
32      ordinary short integer seek. If not, then seekl() first seeks
33      by blocks ( 512 bytes ) and then seeks the rest of the way by
34      bytes.
35
36  CALLS:
37
38      seek() - Unix system call.
39
40  CALLED BY:
41
42      all kinds of routines that read from random locations in large files.
43
44  HISTORY:
45
46      written by Dan Putnam - fall 1978
47
48  *****/
49
50
51
52  seekl(fid, offset)
53  int fid;
54  long offset;
55  {
56      struct          /* for accessing hi and lo words of offset */

```

```
57 {
58     int hi;
59     int lo;
60 };
61
62 register int code;          /* return code from seek
63 register int i;
64
65 if(offset.hi != 0)
66 {
67     if( (code = seek(fid, (i = offset / 512), 3)) < 0)
68     {
69         return(code);
70     }
71     else
72         return(seek(fid, (i = offset % 512), 1));
73 }
74 else
75     return(seek(fid, offset.lo, 0));
76 }
77 }
```

```

1  /*          C O M P A R      (Note: Borrowed from CELDS, Thanks! )
2  * compar two null-terminated strings
3  *
4  * The characters at "s1" and "s2" are compared until one terminates.
5  * If the last characters compared are equal, zero is returned;
6  * if the char from "s1" is > "s2", a positive value is returned,
7  * otherwise a negative value.
8  *
9  * Calls: none
10 * Globals: none
11 * Last modification: 31 mar 77
12 *
13 */
14 int      compar (s1, s2) char   *s1,
15                                *s2;
16 {
17     register char  *p,
18                  *q;
19     register int    greater;
20
21     p = s1;
22     q = s2;
23     while ((greater = *p - *q++) == 0 && *p++ != 0);
24     return (greater);
25 }

```

```

1  #
2  /*****
3
4  NAME:
5
6      bool()
7
8  FUNCTION:
9
10     Perform Boolean operations on files.
11
12  CALLING SEQUENCE:
13
14     int fida
15     int fidb
16     int fidc
17     int opcode
18     int bool()
19
20  PARAMETERS:
21
22     fida      File descriptor of the first argument file.
23
24     fidb      File descriptor of the second argument file.
25
26     fidc      File descriptor of the output file.
27
28     opcode    Specifies the operation to be performed:
29
30                1 - file(a) AND file(b)
31                2 - file(a) OR file(b)
32                3 - file(a) EXCEPT file(b)
33
34  RETURNS:
35
36     Returns the number of items listed in the output file.
37
38  ALGORITHM:
39
40     The input files are read and their contents are used as indices
41     into the "check" array. Bits are set in the "check" array elements
42     to indicate whether a given entry is present in either or both
43     of the input files.
44
45     When the above step is completed, a pass is made through the
46     check array. The index of a "check" array element is written to
47     the output buffer depending on its membership in the input files
48     and the value of the "opcode".
49
50             AND          belongs to file(a) and to file(b).
51             OR           belongs to file(a) or to file(b) or both.
52             EXCEPT     belongs to file(a) but not to file(b).
53
54
55     Note: The "check" array is an array of SPMAX characters, where
56     SPMAX is currently defined at 5000. This parameter should

```

```

57      be large enough for some time to come, and could be set still
58      higher without exceeding core limitations.  However, somebody
59      probably ought to rewrite this routine so that it loops to
60      write the output file in segments.  That is, the routine would
61      make a complete pass through both input files in each iteration.
62      Only those values in the current segment range would be marked
63      in the "check" array.
64
65  CALLS:
66
67      Unix calls:
68
69      seek()
70      read()
71      write()
72
73  CALLED BY:
74
75      select()
76
77  HISTORY:
78
79      written by Dan Putnam - fall 1978
80
81  *****/
82
83
84  #define      AND      1
85  #define      OR      2
86  #define      EXCEPT 3
87  #define      MASKA    01
88  #define      MASKB    010
89
90  #define      SPMAX     5000
91  #define      LSTSIZ    256
92
93  bool(fidb,fidb,fidc,opcode)
94  int fidb;          /* file descriptor of first operand */
95  int fidb;          /* file descriptor of second operand */
96  int fidc;          /* file descriptor of resultant */
97  int opcode;        /* AND, OR or EXCEPT */
98  {
99      char check[SPMAX]; /* check list for membership in lists */
100      int list[LSTSIZ]; /* buffer for input and output lists */
101      register int j; /* fast loop counter */
102      register char *checkptr; /* points thru check array */
103      register int spid; /* id number of source permit in lists */
104      int *listptr; /* points thru list buffer */
105      int *endlist; /* marks end of list array */
106      int maxa; /* max sp id in file a */
107      int maxb; /* max sp id in file b */
108      int maxc; /* upper bound of elements in result */
109      int count; /* for returning size of resultant file */
110
111      /****** first, clear check array */
112

```

```

113
114     checkptr = check;
115     j = SPMAX;
116     do
117         *checkptr++ = 0;
118     while(--j);
119
120
121
122
123     /****** run thru file a checking sp's found in list */
124
125     seek(fida,0,0);          /* make sure we get whole file */
126     maxa = -1;              /* init to find max in file a */
127
128     while( (j = read(fida, list, LSTSIZ * sizeof(*list))) > 0)
129     {
130         j /= 2;              /* j = number of ints read */
131
132         listptr = list;
133
134         while(j-- && (spid = *listptr++) != -1)
135         {
136             maxa = maxa > spid ? maxa : spid;
137
138             check[spid] |= MASKA;
139         }
140         if(spid == -1)
141             break;
142     }
143
144
145     /****** run thru file b checking sp's found in list */
146
147
148     seek(fidb,0,0);          /* make sure we get whole file */
149     maxb = -1;              /* init to find max in file b */
150
151     while( (j = read(fidb, list, LSTSIZ * sizeof(*list))) > 0)
152     {
153         j /= 2;              /* j = number of ints read */
154
155         listptr = list;
156
157         while(j-- && (spid = *listptr++) != -1)
158         {
159             maxb = maxb > spid ? maxb : spid;
160
161             check[spid] |= MASKB;
162         }
163         if(spid == -1)
164             break;
165     }
166
167     /****** now run thru the check array to get output file */
168

```

```

169
170     listptr = list;
171     endlist = list + LSTSI2;
172     checkptr = check;
173     seek(fidc,0,0);          /* start at the beginning of the file */
174     count = 0;              /* init count to zero */
175
176
177
178     switch(opcode)
179     {
180
181         case AND:
182             maxc = maxa < maxb ? maxa : maxb;
183
184             for(j = 0; j <= maxc; j++)
185             {
186                 if( *checkptr++ == (MASKA | MASKB) )
187                 {
188
189                     *listptr++ = j;
190                     count++;
191                     if(listptr >= endlist)
192                         write(fidc,(listptr = list),(endlist - list) * 2);
193                 }
194             }
195             break;          /* end case AND */
196
197         case OR:
198             maxc = maxa > maxb ? maxa : maxb;
199
200             for(j = 0; j <= maxc; j++)
201             {
202                 if( *checkptr++ != 0 )
203                 {
204
205                     *listptr++ = j;
206                     count++;
207                     if(listptr >= endlist)
208                         write(fidc,(listptr = list),(endlist - list) * 2);
209                 }
210             }
211             break;          /* end case OR */
212
213         case EXCEPT:
214             maxc = maxa;
215
216             for(j = 0; j <= maxc; j++)
217             {
218                 /****** if a and not b */
219
220                 if( (*checkptr & MASKA) && !(*checkptr & MASKB) )
221                 {
222
223                     *listptr++ = j;
224                     count++;

```

```
225             if(listptr >= endlist)
226                 write(fidc,(listptr = list),(endlist - list) * 2);
227             }
228             checkptr++;
229         }
230         break;                /* end case EXCEPT */
231     }
232 }
233
234
235
236     /****** terminate list and write out the remainder */
237     *listptr++ = -1;
238
239     write(fidc, list, (listptr - list) * 2);
240
241     return(count);
242 }
243
```

CERL DISTRIBUTION

Chief of Engineers
ATTN: Tech Monitor
ATTN: DAEN-RD
ATTN: DAEN-MP
ATTN: DAEN-ZC
ATTN: DAEN-CW
ATTN: DAEN-RM
ATTN: DAEN-CCP
ATTN: DAEN-ASI-L (2)

US Army Engineer Districts

ATTN: Library

Alaska
Albaton
Albuquerque
Baltimore
Buffalo
Charleston
Chicago
Detroit
Far East
Fort Worth
Galveston
Huntington
Jacksonville
Japan
Jidda
Kansas City
Little Rock
Los Angeles
Louisville
Memphis
Mobile
Nashville
New Orleans
New York
Norfolk
Omaha
Philadelphia
Pittsburgh
Portland
Riyadh
Rock Island
Sacramento
San Francisco
Savannah
Seattle
St. Louis
St. Paul
Tulsa
Vicksburg
Walla Walla
Wilmington

US Army Engineer Divisions

ATTN: Library

Europe
Huntsville
Lower Mississippi Valley
Middle East
Middle East (Rear)
Missouri River
New England
North Atlantic
North Central
North Pacific
Ohio River
Pacific Ocean
South Atlantic
South Pacific
Southwestern

Waterways Experiment Station

ATTN: Library

Cold Regions Research Engineering Lab

ATTN: Library

US Government Printing Office Receiving Section/Depository Copies (2)

Defense Technical Information Center ATTN: DDA (12)

Engineering Societies Library New York, NY

FESA, ATTN: Library

ETL, ATTN: Library

Engr. Studies Center, ATTN: Library

Inst. for Water Res., ATTN: Library

Army Instl. and Major Activities (CONUS)

DARCOM - Dir., Inst., & Svcs.

ATTN: Facilities Engineer

ARRADCOM

Aberdeen Proving Ground
Army Matls. and Mechanics Res. Ctr.
Corpus Christi Army Depot
Harry Diamond Laboratories
Dugway Proving Ground
Jefferson Proving Ground
Fort Monmouth
Letterkenny Army Depot
Natick Research and Dev. Ctr.
New Cumberland Army Depot
Pueblo Army Depot
Red River Army Depot
Redstone Arsenal
Rock Island Arsenal
Savannah Army Depot
Sharpe Army Depot
Seneca Army Depot
Tobyhanna Army Depot
Tooele Army Depot
Watervliet Arsenal
Yuma Proving Ground
White Sands Missile Range

FORS COM

FORS COM Engineer, ATTN: AFEN-FE

ATTN: Facilities Engineers

Fort Buchanan
Fort Bragg
Fort Campbell
Fort Carson
Fort Devens
Fort Drum
Fort Hood
Fort Indiantown Gap
Fort Irwin
Fort Sam Houston
Fort Lewis
Fort McCoy
Fort McPherson
Fort George G. Meade
Fort Ord
Fort Polk
Fort Richardson
Fort Riley
Presidio of San Francisco
Fort Sheridan
Fort Stewart
Fort Wainwright
Vancouver Bks.

TRADOC

HQ, TRADOC, ATTN: ATEN-FE

ATTN: Facilities Engineer

Fort Belvoir
Fort Benning
Fort Bliss
Carlisle Barracks
Fort Chaffee
Fort Dix
Fort Eustis
Fort Gordon
Fort Hamilton
Fort Benjamin Harrison
Fort Jackson
Fort Knox
Fort Leavenworth
Fort Lee
Fort McClellan
Fort Monroe
Fort Rucker
Fort Sill
Fort Leonard Wood

INSCOM - Ch, Instl. Div.

ATTN: Facilities Engineer

Vint Hill Farms Station
Arlington Hall Station

WESTCOM

ATTN: Facilities Engineer

Fort Shafter

MDW

ATTN: Facilities Engineer

Cameron Station
Fort Lesley J. McNair
Fort Myer

HSC

HQ USAHSC, ATTN: HSLO-F

ATTN: Facilities Engineer
Fitzsimons Army Medical Center
Walter Reed Army Medical Center

USACC

ATTN: Facilities Engineer

Fort Huachuca
Fort Ritchie

MTMC

HQ, ATTN: MTMC-SA

ATTN: Facilities Engineer
Oakland Army Base
Bayonne MOT
Sunny Point MOT

US Military Academy

ATTN: Facilities Engineer

USAES, Fort Belvoir, VA

ATTN: FE Mgmt. Br.

ATTN: Const. Mgmt. Br.

ATTN: Engr. Library

Chief Inst. Div., I&SA, Rock Island, IL

USA ARRCOM, ATTN: Dir., Instl & Svc

TARCOM, Fac. Div.

TECOM, ATTN: DRSTE-LG-F

TSARCOM, ATTN: STSAS-F

NARAD COM, ATTN: DRDNA-F

AMMRC, ATTN: DRXMR-WE

HQ, XVIII Airborne Corps and

Ft. Bragg

ATTN: AFZA-FE-EF

HQ, 7th Army Training Command

ATTN: AETTG-DEH (5)

HQ USAREUR and 7th Army

ODCS/Engineer

ATTN: AEAEN-EH (4)

V Corps

ATTN: AETVDEH (5)

VII Corps

ATTN: AETSDEH (5)

21st Support Command

ATTN: AEREH (5)

US Army Berlin

ATTN: AEBA-EN (2)

US Army Southern European Task Force

ATTN: AESE-ENG (5)

US Army Installation Support Activity,

Europe

ATTN: AEUES-RP

8th USA, Korea

ATTN: EAFE

Cdr, Fac Engr Act (8)

AFE, Yongsan Area

AFE, 20 Inf Div

AFE, Area II Spt Det

AFE, Cp Humphreys

AFE, Pusan

AFE, Taegu

DLA ATTN: DLA-WI

USA Japan (USARJ)

Ch, FE Div, AJEN-FE

Fac Engr (Honshu)

Fac Engr (Okinawa)

ROK/US Combined Forces Command

ATTN: EUSA-HHC-CFC/Engr

INS Branch Distribution

Picatinny Arsenal
ATTN: SMDPA-VP3

Directorate of Facilities Engr
Miami, FL 34004

DARCOM STIT-11R
AIM New York 09110

West Point, NY 10996
ATTN: Dept of Mechanics
ATTN: Library

HQDA (SGRD-EDE)

Chief of Engineers
ATTN: DAEN-MPO-B
ATTN: DAEN-MPR
ATTN: DAEN-MPO-U
ATTN: DAEN-MPZ-A
ATTN: DAEN-RDL
ATTN: DAEN-ZCE

National Defense Headquarters
Director General of Construction
Ottawa, Ontario K1A0K2
Canada

Airports and Const. Services Dir.
Technical Information Reference
Centre
KAIL, Transport Canada Building
Place de Ville,
Ottawa, Ontario K1A0N8
Canada

British Liaison Officer (5)
U.S. Army Mobility Equipment
Research and Dev Center
Ft. Belvoir, VA 22060

Aberdeen Proving Ground, MD 21005
ATTN: AMXHE/J. D. Weiss

Ft. Belvoir, VA 22060
ATTN: Learning Resources Center
ATTN: ATSE-TD-TL (2)
ATTN: Kingman Bldg, Library
ATTN: MAJ Shurb (4)

Ft. Leavenworth, KS 66027
ATZLCA-SA/F. Wolcott

Ft. Monroe, VA 23651
ATTN: ATEN-AD (3)
ATTN: ATEN-FE-E

Ft. Lee, VA 23801
ATTN: DRXMC-D (2)

HQ FORSCOM
ATTN: AFEN-CD
Ft. McPherson, GA 30330

5th US Army
ATTN: AKFB-LG-E

6th US Army
ATTN: AFKC-EN

US Army Engineer District
New York
ATTN: Chief, NANEN-E
ATTN: Chief, Design Br.
Pittsburgh
ATTN: Chief, Engr Div
Philadelphia
ATTN: Chief, NAPEN-E
Baltimore
ATTN: Chief, Engr Div
Norfolk
ATTN: Chief, NAOEN-R
Huntington
ATTN: Chief, ORHED-P

US Army Engineer District
Wilmington
ATTN: Chief, SAWEN-PP
ATTN: Chief, SAWEN-PM
ATTN: Chief, SAWEN-E

Charleston
ATTN: Chief, Engr Div
Savannah
ATTN: Chief, SASAS-L
Jacksonville
ATTN: Env. Res. Br.
Nashville
ATTN: Chief, ORNED-P
Memphis
ATTN: Chief, LMED-PR
Vicksburg
ATTN: Chief, Engr Div
Louisville
ATTN: Chief, Engr Div
St. Paul
ATTN: Chief, ED-ER
Chicago
ATTN: Chief, NCCPD-ER
St. Louis
ATTN: Chief, ED-B
Kansas City
ATTN: Chief, Engr Div
Omaha
ATTN: Chief, Engr Div
Little Rock
ATTN: Chief, Engr Div
Tulsa
ATTN: Chief, Engr Div
Fort Worth
ATTN: Chief, SMFED-PR
ATTN: Chief, SMFED-F
Galveston
ATTN: Chief, SWGAS-L
ATTN: Chief, SWGCO-M
Albuquerque
ATTN: Chief, Engr Div
Los Angeles
ATTN: Chief, SPLED-E
San Francisco
ATTN: Chief, Engr Div
Sacramento
ATTN: Chief, SPKED-D
Far East
ATTN: Chief, Engr Div
Seattle
ATTN: Chief, NPSEN-PL-WC
ATTN: Chief, NPSEN-PL-ER
ATTN: Chief, NPSEN-PL-BP
Walla Walla
ATTN: Chief, Engr Div
Alaska
ATTN: Chief, NPASA-R

US Army Engineer Division
New England
ATTN: Laboratory
ATTN: Chief, NEDED-E
South Atlantic
ATTN: Chief, SADEN-E
Huntsville
ATTN: Chief, HNDED-CS
ATTN: Chief, HNDED-M
Lower Mississippi Valley
ATTN: Chief, PD-R
Ohio River
ATTN: Chief, Engr Div
North Central
ATTN: Chief, Engr Planning Br.
Southwestern
ATTN: Chief, SWDCO-O
South Pacific
ATTN: Laboratory
Pacific Ocean
ATTN: Chief, Engr Div
ATTN: Chief, PODED-P
North Pacific
ATTN: Laboratory
ATTN: Chief, Engr Div

McClellan AFB, CA 95652
2857 APG/DE (Lt David C. Hall)

Peterson AFB, CO 80914
HQ ADCOM/DEMUS (M. J. Kerby)

Tinker AFB, OK 73145
2854 ABG/DEE (John Wall)

Patrick AFB, FL 32925
Base CE Sqdn (James T. Burns)

AF/ROXT
WASH DC 20330

AFESC/PRT
Tyndall AFB, FL 32403

Little Rock AFB
ATTN: 314/DEE (Mr. Gillham)

Kirtland AFB, NM 87117
ATTN: DEP

US Naval Oceanographic Office
ATTN: Library
Bay St. Louis, MS 39522

Naval Facilities Engr Command
ATTN: Code 04
Alexandria, VA 22332

Port Hueneme, CA 93043
ATTN: Library (Code L08A)
ATTN: Morell Library

Washington, DC
ATTN: Building Research Advisory Board
ATTN: Transportation Research Board
ATTN: Library of Congress (2)
ATTN: Dept. of Transportation Library

Dept of Transportation
ATTN: W. N. Lofroos, P. E.
Tallahassee, FL 32304

LT Neil B. Hall, CEC, USNM (Code 100)
884-6366
US Navy Public Works Center
Box 6, FPO San Francisco 96651

HQ AFESC
ATTN: DEVP/CPT R. Hawkins (100)
ATTN: TST/Library (2)
Tyndall AFB, FL 32403

Webster, Ronald Dwight

Interagency/Intergovernmental Coordination for Environmental Planning (IICEP):
systems considerations / by R. D. Webster, D. E. Putnam. -- Champaign, IL : Con-
struction Engineering Research Laboratory ; Springfield, VA : available from NTIS,
1980.

101 p.; 27 cm. (Technical report ; N-87)

1. State governments -- directories-data processing. 2. Environmental policy-
directories-data processing. 3. U.S. Air Force-environmental aspects. I. Putnam,
Daniel E. II. Title. III. Series: U.S. Army Construction Engineering Research
Laboratory. Technical report ; N-87.

