RESEARCH REPORT

SCEINTIFIC AND TECHNICAL INFORMATION TRANSFER FOR EDUCATION (STITE)

FINAL REPORT

Pranas Zunde
Project Director

June, 1976

GEORGIA INSTITUTE OF TECHNOLOGY
ATLANTA, GEORGIA

RESEARCH REPORT

SCEINTIFIC AND TECHNICAL INFORMATION TRANSFER FOR EDUCATION (STITE)

FINAL REPORT

Pranas Zunde
Project Director

June, 1976

GEORGIA INSTITUTE OF TECHNOLOGY

ATLANTA, GEORGIA

SCIENTIFIC AND TECHNICAL INFORMATION TRANSFER FOR EDUCATION

(STITE)

Final report on research performed at the School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia, under the National Science Foundation Grant No. GN-36114.

ABSTRACT

While the utilization of existing science and technical information systems has, in the past, been primarily in research, it is desirable that the use of such valuable resources be extended to other areas of intellectual endeavor. Science education, because of its inherent function of transferring information from an external source into the human mind, is a natural candidate for the extended use of these centers and systems. The recent development of technology-aided learning systems which permit learners to interact with organized learning materials stored in an inanimate manipulable device, or memory, strengthens the possibility of an increased utilization of science information for the purposes of instruction and learning.

The main objective of the STITE project was to study man-machine mechanisms for enhancing the transfer of science information from its present repositories into science learning systems. More specifically, the research had the following goals:

1. To describe operationally the human process of transformation of science information system outputs for the purpose of integrating them into learning systems;
2. To investigate comparatively the design and operating characteristics of science information systems and science learning systems, particularly from the viewpoint of requirements for transferring information between them via a man-machine interface;
3. To implement an experimental design of an interface system and a limited transfer mechanism and to study selected aspects of that mechanism.

In the course of this research, the above goals were successfully implemented. In particular, selected science and technical information systems were analyzed and information acquisition habits and needs of science educators were thoroughly studied. Subsequently, an experimental

interface mechanism was developed which helps the educator to structure selected types of tasks characteristic for this profession, such as to make better use of existing information resources. The idea of the interface system is to store in the system certain amounts of information in highly structured modular form, relevant to specific needs of various groups of educators. The modules in the system are manipulable and can be assembled into various sequences geared to support various tasks which educators are expected to perform in their profession. The essential feature of such a modular interface system, which provided the basis for the design of an experimental program, is the interaction between the stored internal information about the subject matter and the external information from the information repository.

Demonstrations of the system were made on the campus of the Institute, with graph theory chosen as the experimental subject area.

PREFACE

This report consists of two main parts: a summary of all the research work on scientific and technical information transfer for education, called STITE Project, and a detailed description of work performed in this final period of that project (see in particular Appendices). Persons interested in a detailed description of the research work performed in the earlier phases of the STITE Project are referred to the four earlier reports published in the years 1973 - 1975 (bibliographic data on those reports can be found on page 47 of this report).

In addition to various analytical studies made under the STITE Project, the major product of this research effort was the development of an experimental interface system designed to enhance the use of scientific and technical information by educators. The experimental system is in working condition and was demonstrated, on several occasions, to the Georgia Tech faculty and students and also to the NSF Project Officer, Dr. Edward C. Weiss. It is, however, true that the system needs yet to be evaluated and tested under various operational modes. Its actual application, in the present state, is furthermore limited by existing hardware capacities and the subject area selection. It is hoped that some of the tasks of evaluation and testing as well as upgrading of the software and hardware of the experimental system can be accomplished in the future.

In the last phase of the research, reported in this document, the main contributor to the STITE Project was Dr. L.J. Gallaher, Senior Research Scientist, who designed and developed the software of the experimental system. His contribution in this and earlier phases is duly acknowledged. We also gratefully acknowledge the contributions of other faculty members and graduate students of the School of Information and Computer Science of the Georgia Institute of Technology, who participated in various phases of the STITE

Project and whose talent and dedication made this project a success. They are: faculty members, Dr. Miroslav Valach, Professor, ICS; Dr. T.C. Ting, Associate Professor, ICS; Dr. Albert N. Badre, Assistant Professor, ICS; and Mrs. Dorothy S. Hughes, Research Analyst; and the graduate students, Mr. Frank Chu, Khalid Hafiz and N.V. Subramanian. Last but not least we duly acknowledge the moral and technical support and the expert advice of the Director of the School of Information and Computer Science, Dr. Vladimir Slamecka, which we were priviledged to enjoy from the very inception of this project.

This research was made possible by the National Science Foundation under grant No. GN-36114, and the Foundation's interest in and support of this work as well as the kind cooperation of its Project Officer, Dr. Edward C. Weiss, is sincerely appreciated by all who participated in this endeavor.

Pranas Zunde
Project Director

TABLE OF CONTENTS

I. OBJECTIVES AND SCOPE OF RESEARCH

I.1 INTRODUCTION

Research and development in the field of science information during the past decade has resulted in the establishment of large banks of descriptive information and bibliographic data. Stored on digital and analog media, these collections, along with the organization and dissemination of their data and information, comprise science and technical information centers, or science information systems.

So far, the almost exclusive clientele of science information systems has been the community concerned with research, development and production of either ideas or goods. It is logical that the science and technical research community be considered as the principal beneficiary of science information systems: the impetus for the establishment of these systems was given by concern with the efficiency of scientific research and development in a society in which both the quantity of science information and the number and variety of its users have registered a substantial growth trend. At the same time, however, the scientific research community as a market is not characterized by either the volume or the frequency of information use which would exhaust the capacity and potential of present-day science information systems. It is thus very appropriate to inquire whether the contents and services of these systems can be made available to science endeavors other than research and development. The appropriateness of this inquiry is unquestionable; science information is a social and national resource whose value potential is closely related to the level of its prudent use.

In searching for other markets for science information, the obvious direction is to look toward human activities which are heavily dependent on information inputs from external "science memories."

Obviously, human science learning is an example of such an activity which fundamentally depends on information transfer from external information sources into the human mind. Science education, the organized social system for such learning, is intuitively an attractive market for science information.

The notion of a productive interplay between science information and learning is not new, although until very recently it has received little overt attention. Watson Davis (5) is reported having observed "a curious lack of follow-up among documentalists" to the utopian idea of a "world brain" suggested by H.G. Wells—a centralized store of knowledge used for education. More recently, the Interuniversity Communications Council proposed to bring the contents of various existing and potential information stores to bear on the educational process; after an initial period of enthusiasm (7), the concept still remains to be seriously explored, as do the premises which led to the "networks for knowledge" Federal legislation of the mid-1960's. Apart from a few serious advocates of such a partnership (8), the respective orientations of science information and education have remained diverse, even on occasions when the two met to communicate on a shared platform of educational technology.

Recently, however, several events are indicative of a high level concern with the relationship of science information and education. Prominent among these is a study by the Organization for Economic Cooperation and Development entitled Information for a Changing Society (OECD, 1971). In response to a major conclusion of this study (that "information systems of the future must be dynamic, capable of educating, and adaptable to the changing educational systems of the world") OECD recommends that "evaluation of educational requirements of modern societies should take account of the need for information transfer systems that are better adapted to the continuing re-education of adults."

In a parallel vein, specific attention has now been drawn to the necessity of adapting science information systems to the science educational process and system of the future: "The information systems stimulated by the National Science Foundation have heretofore focused primarily on the needs for specialists; greater attention should be paid to the life-long educational process of the [scientific] non-specialists" (6).

I.2 Science Information and Education

In the context of the work reported here, science information is defined as a recorded knowledge, usually in the form of primary publications stored on analog or digital media, emanating from the intellectual work of the scientific and technical community. The management of the inventory of science information, and the enhancement of its utility to mankind, are the primary purposes of science information systems. In the attempt to meet these purposes, science information systems collect, evaluate, organize and store science information, and they provide tools for its purposeful use. In modern science information systems, the tools comprise primarily devices and mechanisms for accessing science information—devices and mechanisms such as descriptive notations, indices, abstracts, searching methods, etc. The function of science information systems may be said to be that of a couple between science information and its users, insofar as present-day systems are generally limited only to the storage of science information surrogates.

In their role of searchable bibliographic directories, the use of science information systems is not limited to the research community; their use by others depends, however, on their need for science information and on its utility. The use of science information systems in science education has been sporadic, and largely confined to individuals engaged in research rather than in scientific or technical instruction or learning. It is quite apparent that the flow of science information into science education is characterized by discontinuity, lack of intensity, and time delay; hence its utilization in science education is suboptimal.

Unless the use of science information systems in science education is to be for purposes other than an occasional bibliographic search on a topic of momentary interest, the possibility of an effective flow of science information into education appears to be predicated on the existence, availability and use of external, manipulable stores of learning materials. A direct flow of science information from its repositories into the highly volatile and transitive environment of live classroom instruction is difficult to imagine; the updating of the knowledge of human learners is an idiosyncratic process of habitual characteristics which are not easily modified. On the other hand, stored learning materials (textbooks, instructional films, etc.) are a solid client of science information services, and a strong element in the educational process.

The realistic possibility of an intensified influx of science information into instruction and learning in the sciences is therefore given by the recent emergence and development of "learning systems," broadly defined as technology-aided instruction/learning facilities which allow learners to interact with organized learning materials stored in an animate, manipulable device or memory. The more powerful of these systems are "conversational"—that is, interactive systems which provide, in addition to a modular store of learning materials, models of the live interaction between a student and a human tutor. Despite the relative simplicity of the present-day models, the conversational learning systems hold promise of being able to sustain realistically the process of self-instruction. Furthermore, given certain types of memories of learning materials to support the self-instruction process, it is possible to consider science information as constituting an important input into these systems.

I.3 Research Objectives

The objective of this projectwas to study, design and experimentally evaluate man-machine mechanisms for enhancing the transfer of science information from its present repositories into science learning systems.

The postulated mechanism for the transfer of science information into learning systems is illustrated in Figure 1. The main characteristic of this transfer mechanism is the use of certain outputs of science information systems, enhanced and modified as appropriate, as inputs to the learning systems. The transformation of the outputs, and the ease and economy with which the transfer can be accomplished, are the crucial aspects of the mechanism. It is unlikely that the process can be performed fully automatically; rather, the educator-author is the key human interface responsible for the transformation of information.

Within the general objective of investigating the mechanisms of science information transfer into science education, the proposed research had the following specific goals:

1. To describe operationally the human process of transformation of science information system outputs for the purpose of integrating them into learning sytems;

2. To investigate comparatively the design and operating characteristics of science information systems and science learning systems, particularly from the viewpoint of requirements for transferring information between them via a man-machine interface;

3. To implement an experimental design of a man-machine interface system and a limited transfer mechanism from appropriate existing science information systems for instructional purposes, and to evaluate selected aspects of that system.

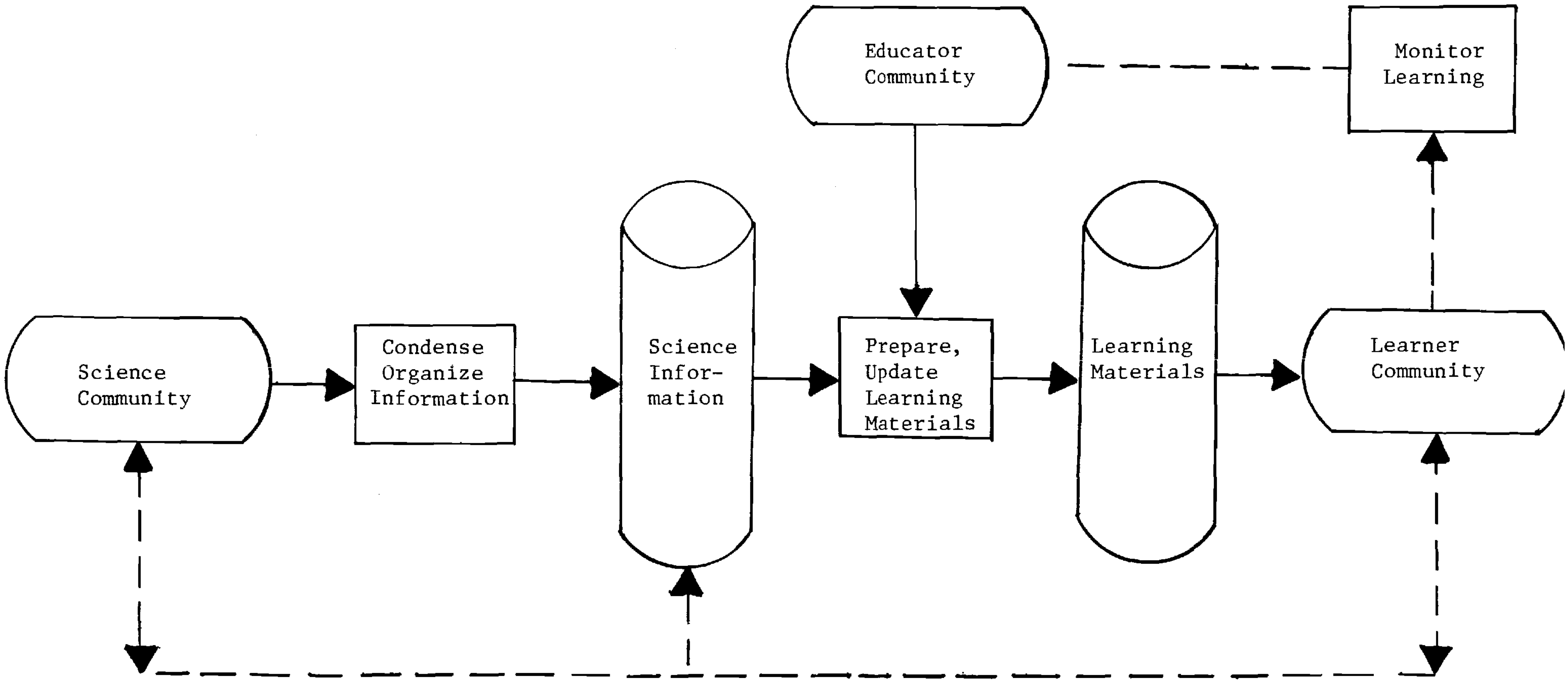


FIG. 1 Utilization of Science Information In Learning Systems

II LITERATURE SURVEY

The design of the STITE system was preceded by a review of the research literature on the systems approach to curriculum development. The purpose of this review was to identify and evaluate developments and research that have already been conducted in this field and to determine their usefulness in the design of the STITE system.

The subject range of the project was delineated in terms of topics and categories, and then appropriate indexes, reports, and other publications were searched to identify and evaluate those materials which are significant for the goals of this project. The literature survey was organized in three major subject matter categories, which reflect the three major thrusts of the project.

First, there was an identification and an analysis of existing scientific and technical information centers and systems in terms of the subjects they include, their operating characteristics and the ways in which they are used.

Second, there followed a definition and description of learning tools and systems, and an analysis of the methodology involved in designing them, as well as some evaluation of their usefulness with respect to the objectives of this project.

Third, there was the consideration of information transfer systems and networks to determine their significance for the design of the proposed transfer system.

The subject scope of the literature search followed the three aforementioned divisions, i.e., science and technical information centers, learning systems, and information transfer systems and networks. Initially, spot checks for relevant references were made within the designated subject areas, and this preliminary and tentative outline of the survey, with some random references, has been developed. This was then expanded and refined as the project developed.

The first and second STITE Progress Reports (1,2) contain selected bibliographies on the above mentioned topics together with a discussion of their relevance.

III PRELIMINARY CONSIDERATION AND INVESTIGATION

III.1 Analysis of Selected Science and Technical Information Centers

The prominent products of research and development in science/technical information of the past decade are large banks of descriptive information and bibliographic data stored on digital and analog media. The collection, organization, and dissemination of this information and data comprises the so-called "Science and Technical Information Systems/Centers" (STIC), whose services typify the current level of development of the information industry - the establishment of computerized information utilities and services of different types of purposes. Science and technical information centers differ widely when considered from the viewpoint of the degree of automation, the depth and breadth of subject coverage, their sources of information, their operational characteristics and their availability to the user community. By user community is meant the total user population of the center.

Figure 2 shows the information flow in the proposed STITE system. The main aim of this research project is the design of this central system.

The STITE system is conceived as an intermediary office between the LIS and STICs with computing and data processing facilities. On the basis of requests from the educator community (i.e. LIS), the STITE system would interact with STICs according to some pre-programmed fashion and according to its intended purpose, it would acquire the necessary information from the STICs and provide the same to the appropriate LIS. The selection of media for information transmission between the components of the proposed system, as shown in the above figure, is one of the main features of the design of the proposed information transfer mechanism. It can be anticipated that there would be a gradual transition from off-line to on-line, and from batch to real-time transmission. As technology advances and

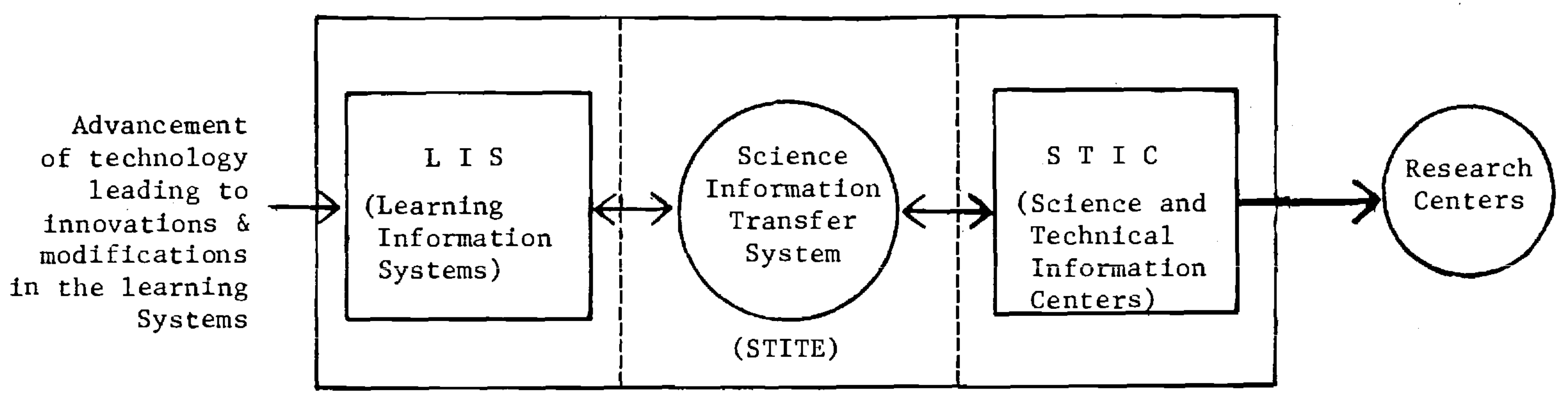

Advancement
of technology
leading to
innovations &
modifications
in the learning
Systems
L I S
(Learning
Information
Systems)
Science
Information
Transfer
System
(STITE)
S T I C
(Science and
Technical
Information
Centers)
Research
Centers

Fig. 2

brings down the cost of equipment, such as audio-visual equipment and data transmission devices, there will be innovations and modifications in learning information systems (LIS). This will continually alter the information requirements of LIS, and consequently its interactions with the STITE system will enhance with time.

A few basic assumptions were made concerning science information systems and the design philosophy of the proposed science information transfer system.

1) The science information centers will have direct contact, or contact via an intermediary office, with the user community on an initiative as well as on a responsive basis.

2) Not all the learning systems will required the transfer of information from science information systems to learning systems. This variation will necessitate further research defining the criteria for the use of STITE system and identifying the courses that need updating with the science information to be transferred from these centers.

3) There may be an additional performance requirement on these science information centers to publish periodicals that will assist the educator community in the development of new courses to be offered to learners for enhancing their knowledge with current research results.

4) It was further assumed that to determine the science information centers to be accessed for the development of new courses in a given subject area and to facilitate the identification of centers that will provide the necessary science information for updating the learning materials for modifying the course preparation in a given subject area, it is necessary to categorize these centers on the basis of subject disciplines.

The distinctions between different science information systems relate to the fact that there are various methods for each of the stages of operation, such as acquisition, record keeping, library processing and loans, storage, information dissemination, information searching, information retrieval, etc.

For the purpose of this project, science and technical information systems were analyzed as information service systems which operate in an environment which includes both information to be processed and population of users who require information in order to achieve certain goals. Consequently, existing science and technical information centers were first categorized on the basis of subject disciplines. Within that categorization, information centers were analyzed in terms of activities, scope or subject coverage, input sources, holdings of recorded data, microform services, and other services provided by the centers such as abstracting, indexing, literature searching, and data collection and analysis.

The analysis of the characteristics of information processed by the centers then focused on the functional categorization of information into (i) initiative outputs and services and (ii) responsive outputs and services which have potential benefits for educators. In particular, it was investigated how those outputs can help the educator community in improving and further developing various teaching tools and techniques. The results of this survey and analysis of existing scientific and technical information centers are reported in (9).

III.2 Description of Learning Information Systems

Another major task of the STITE project was to obtain a pragmatic understanding of the process employed by science educators in developing and structuring science learning materials. The following aspects of the existing learning systems and instructional processes were studied in some detail.

1. Analysis of the "socratic" principles in the teaching-learning process.

2. Analysis of the process of developing conventional teaching materials (course preparation, course development).

3. Description and design of programmed instructional materials.

4. Analysis of computer assisted instruction.

5. Analysis of audio-visual techniques.

6. Analysis of self-instructional systems.

The first STITE Progress Report (1) gives a thorough discussion of each of these points and concludes with a tentative matrix showing the feasibility of the use of the outputs of the existing information centers as inputs to the learning system. It should be noted, however, that a more detailed analysis is required (classification by subject matter, level of difficulty, etc.) before the potential information requirements for new educational tools can be given.

III.3 Information Acquisition Practices and Needs of Science Educators

III.3.1 INTRODUCTION

As it was already mentioned, the goal of the STITE project was to implement an experimental mechanism for the purpose of transferring scientific and technical information from its present repositories into a system that will enhance the use of scientific and technical information centers by providing that information in a form and manner that corresponds more precisely to the exact needs of the science educator community.

One prerequisite for the achievement of this goal was some understanding of the actual information needs of science educators and the factors, external to the information itself, that influence the educator in the acquisition and use of realistic conclusions about the kind of information that educators required for their particular tasks and the conditions that influence its acquisition and use.

STITE followed two lines of inquiry to answer these questions. One was directed at the whole population of educators at the institutions of higher learning. A literature analysis investigated previous studies about the needs of the particular group of information users in which STITE is interested, the science educators. A questionnaire sent to science educators at various colleges and universities over the country solicited answers from science educators themselves about their use of existing information centers, the types of information they use, for what purposes they use it, and the features that they feel might increase their use of information sources.

The second line of inquiry in surveying the informational needs of science educators was directed toward a group of actual users of a scientific and technical information center, and namely the group of users of the Georgia Information Dissemination Center operated by the University of Georgia in Athens, Georgia.

Data received in response to both of these inquiries was very useful in the design of the experimental STITE system.

III.3.2 Survey of Informational Needs of Science Educators

While user studies reveal some patterns of information need and use by science educators, the interest of STITE was centered primarily on the utilization of existing information centers. Further investigation into the actual use of these centers by science educators was considered desirable. This additional effort was made through a questionnaire survey aimed specifically at science educators in various American universities.

The literature survey indicated that, at present, the main utilization of most scientific and technical information systems can be traced to the industrial and research communities, and that the use of such information systems for educational purposes is probably minimal. Accordingly, the purpose of the questionnaire was to determine to what extent science educators indeed make use of science and technical information centers as well as under what conditions and for what reasons they do so. It was also hoped that one of the important by-products of this inquiry would be the determination—more precise than this has been done in previous studies—of the information needs of science educators.

The results of the study convincingly demonstrated (1) that a large percentage of science educators are not aware of the availability or of the existence of scientific and technical information systems and (2) that most science educators who are aware of the availability of scientific and technical information systems have no ready or easy access to them. The percentage of educators which are not aware of the existence of such systems ranges from 71% in social sciences to 56% in the physical sciences, with 61% the overall average. And although 80% of educators who are aware of such systems also make actual use of their services, only 53% of the users of the systems have direct access to them. In general, the study indicated that exposure of information services to science educators must be considered an important factor in the improvement of the use of information centers (STITE Progress Report No. 3 [Ref 3], Badre, Hughes, Ting, and Zunde, 1974 [Ref 9]).

III.3.3 Analysis of the Educational Use of an Information Center

Results of the survey of the educational needs of science educators seemed to indicate that further investigation into the instructional usage of a specific information center might yield some helpful results, and a group of users of the Georgia Information Dissemination Center was selected by STITE for this inquiry.

The Georgia Information Dissemination Center, or GIDC, is a bibliographic retrieval system serving the faculty, research staff, and graduate students of the University System of Georgia. It began in 1968 and searches multiple data bases to provide both SDI and retrospective search services to its users. Physical facilities are located on the university campus in Athens, and remote users from Georgia's eleven other senior colleges and twelve junior colleges are usually served through reference librarians on those campuses.

A questionnaire was sent to 1,211 users of the Center, of which 410, or almost 33%, were answered. In this questionnaire, comments of the users were solicited on specific educational uses of services of the Center, subject areas of utilization, sufficiency of information in the surrogate form, possible improvements in the systems, etc.

One hundred eighty-eight persons, or 45.8%, indicated that the material they received from GIDC had been used for some type of instructional purpose, and Question 3 brought out the fact that the most common instructional use is the compilation of bibliographies or reading lists. This type of use was indicated by 137 users, or 33.4%. Utilization for current awareness in the subject area of a course was indicated by 98 respondents (23.9%), while 80 (19.5%) said that they used the information for collection of data. Updating an existing course was a utilization cited by 62 (15.1%), while the tasks of preparation of illustrative examples and development of new courses were indicated by 42 (10.2%) and 34 (8.2%) users respectively. The "other" category checked by 34 persons most often referred to directing research projects of students and developing research papers. Selection of case studies and preparation of quizzes and tests were less frequent uses.

Most of the educational users (136 or 72.2%) indicated that, for their purposes, it was necessary for them to obtain full-text documents rather than to rely on titles and abstracts, although 81 did say that titles and abstracts are sufficient. In a few cases there seemed to be some confusion about the availability of abstracts and methods of securing them.

Of the suggested possible changes and improvements, the following were favored by the users in order of preference.

More descriptive abstracts	39.2% or 161 users
Browsing capability	29.5% or 121 users
Interactive system for query or profile formulation	27.3% or 112 users
Easier access to the system	22.4% or 93 users
Shorter waiting time for information delivery	22.4% or 92 users

Based upon the information needs of science educators as they are reflected in user studies and surveys, it would seem that an information system should revolve around easy and rapid access to materials that are useful either for the purpose of keeping up-to-date or for answering particular questions with specific data. Low motivation and lack of time on the part of the science educator set the requirement for ease of use and rapid retrieval; definitions, descriptions of processes, and outlines are the kinds of specific information that he requires. Concerning keeping up-to-date, his need is for representative, not necessarily comprehensive, information on given topics with perhaps also some evaluation of materials. In this respect, a well-developed review literature would seem to be of great value to the science educator for it could present selections of up-to-date information on given topics without requiring a great amount of time for reading and assimilation. For keeping up-to-date, capitalizing on the advantages and popularity of informal sources of information, such as conversations with colleagues, could involve the inclusion in the information system of lists of authorities in a particular field with notes on their activities.

Some bibliographic tools for retrospective searching are needed but here again the requirement is for representative, rather than comprehensive, references. For details, see STITE Progress Report No. 4 (4).

IV CONCEPTS AND FUNCTIONS OF THE SYSTEM

IV.1 Major Functions of the STITE System

IV.1.1 Introduction

One of the major areas of concentration of research effort was the identification and specification of the types of potential outputs of the STITE system which could be helpful to educators in preparing, developing, and maintaining teaching materials and devices of all sorts. Seventeen classes of such hypothetical outputs were identified; in the sequel, a class of outputs of the STITE system, associated with some particular function, is referred to as task. For each of these tasks, inputs required for implementing the particular task, type of system's output, options available to educators in modifying tasks and data base requirements for a particular task were determined. All these specifications were prepared primarily with graph theory as the experimental subject area in mind.

IV.1.2 List of Tasks

1. Task No. 1. - Preparation of Course Outline

2. Task No. 2. - Preparation of Narrative Presentations

3. Task No. 3. - Presentation of illustrative examples (graphic data, or a table or a numerical example) for theorems. Illustrative examples for concepts. Here the examples need to be listed under each sub-topic, each theorem, and each concept.

4. Task No. 4. - Presentation of Problems Under Each Sub-topic or Under Each Theorem Sub-topic or Under Each Concept in a Sub-topic.

5. Task No. 5. - Presentation of Bibliography Under Each Topic or Sub-topic.

6. Task No. 6. - Presentation of Application of Theoretical Concepts in Each Major Topic.

7. Task No. 7. - Presentation of Case Studies (i.e. citation of real world cases and analysis of the same from the point of view of some of the theoretical concepts explicated in the earlier tasks. For example, in the field of graph theory they might include engineering applications, human sciences applications, application of physical sciences, operations research, etc.)

8. Task No. 8. - Presentation of Abstracts of Specific Documents.

9. Task No. 9. - Presentation of the State-of-the-Art or of Abstracts Covering a Specified Period.

10. Task No. 10.- Presentation of Historical Development of a Specific Discipline (i.e. retrieval by date of publication so that latest is retrieved last.)

11. Task No. 11.- Preparation of Quizzes of Tests for Students.

12. Task No. 12.- Presentation of a Set of Relevant Questions on Topics.

13. Task No. 13.- Identification of New Concepts and Preparation of Updated Materials for a Specific Topic.

14. Task No. 14.- Retrieval of All Abstracts Related to a Specific Abstract.

15. Task No. 15.- Presentation of Definitions of Concepts. (i.e. given an explanation or definition of a concept present the explanations or definitions of all the concepts required for the explanation of the specific concept under consideration.)

16. Task No. 16.- Presentation of Subjects, With References and Abstracts for Assignments for Students.

17. Task No. 17.- Presentation of a Measure of Newness of a Particular Concept. (Given a statement or concept as input, provide a measure of newness of the concept on processing the descriptors or the keywords in the new concept, comparing it with the descriptors of the concepts in store.)

IV.2 Interactive STITE Interface System for Effective Use of Information Stores

As noted earlier, existing science and technical information systems have limited utility for educators because, among other things, they do not allow the user to shape the interaction with those systems to fit his needs and because the existing search mechanisms do not provide directions for further search. Although these are not the only factors which limit the usefulness of science and technical information systems to educators, they seem to be certainly of great importance. Hence, one of the goals here was to develop, as a part of the STITE system, an interface mechanism which would help the educator to structure selected types of tasks which are characteristic for this profession and which would induce him, through various automated procedures, to make better use of existing information resources.

The underlying idea of the proposed interface system is to store in the system certain amounts of information in highly structured modular form, relevant to specific needs of various groups of educators. The modules in the system are manipulable and can be assembled into various sequences geared to support various tasks which educators are expected to perform in their profession.

The essential features of such a modular interface system, which provided the basis for the design of a small experimental program, are summarized below. The subject matter of the experimental design has been somewhat arbitrarily limited to graph theory.

A module has been defined as a manipulable unit of information of which outputs of the STITE system are constructed. Usually a module will describe, display, illustrate or define a single object or entity such as a thing, event, concept, relation, or anything else to which attention can be directed.

In a broad sense, a module is comparable to a record of a file system. Its length or size might vary depending on the nature of discourse, context, type of materials, etc.

The modules are assigned codes to distinguish them according to (1) the form in which material is presented in the module, (2) the type of material presented, and (3) the level of difficulty of the material contained in the module.

A complete and detailed description of these modules are given in the third STITE Progress Report (3), together with many example modules. A briefer description of the modules is given (section IV.3.4) of this report dealing with file structure.

IV.3 The Design and Operation of STITE Data Base Management System

IV.3.1 Introduction

As already mentioned, the major functions of the STITE system were broken down into tasks,which are discussed in the second STITE Progress Report (2). All of these tasks center around the use of the "internal" information, structured as modules and described in the third STITE Progress Report (3).

The main function of such modular "internal" information is to enhance systems applications, particularly in retrieval of "external" information. This external information, stored in various scientific and technical information centers, consists normally of standard bibliographic data, keywords used to index documents, and in some cases, abstracts. Henceforth this information will simply be called "external data".

This section will describe programs which have been developed for STITE data base management, including both internally and externally retrieved information. In particular, the physical and logical structure of the files, together with a command language for manipulating the data structures in the files, and the interface systems are discussed.

IV.3.2 Data Base Elements

Data base elements are records of two kinds: modules of internal information and units of external information, the bibliographic records.

As described in detail elsewhere, each internal modules consist of textual material and additional information about the textual material (3).

Information associated with text is arranged into fields, with the principal or main topic being designated as A field. B field contains a set of terms which are used to describe or to explicate the principal topic, or A field. C field indicates the source document from which the textual material is taken, F field specifies the form of the text (i.e., definition, theorem, or problem), and the D field refers to the level of difficulty of the textual material.

Modules can be linked to each other by various relationships defined on these fields, and structures can be built of modules or their components. It is these structures or relationships that are of prime interest and that were used to aid in selecting the bibliographic information.

In an operational system of this kind, external data would be supplied by an on-line link to various data bases, such as the DIALOG (Lockheed) system or the Georgia Information Dissemination Center. However, for demonstration purposes, these external sources were simulated by sample data; an actual on-line link was not economically feasible within the scope of the financial resources of the STITE project.

It was postulated that external data would be processed by the STITE system in two steps, or levels. At the first level, external data would be retrieved by queries submitted to the external source in the standard format expected by that source. At the second level, information retrieved at the first level would be processed for the purposes of achieving compatability with the records of the internal system, extracting additional information, and updating lists and file structures. Thus, at the first level, the main task can be viewed as presenting an intelligent request to external sources using the internal information stored in the modules, whereas at the second level, the main task can be seen as casting the records retrieved from external sources into a format approaching as closely as possible the format of complete modules.

IV.3.3 General Description of the STITE Interface Systems

Figures 3 through 6 are diagrams displaying main features of the STITE User's System. These diagrams represent the system from the point of view of the user and emphasize the interaction with the user.

Information about the system, its capabilities, and the type and extent of its information stores is incorporated into the system and is supplied to the user by the system in a dialogue form.

Explanations to the user about the system are of several kinds and constitute an on line "user's manual". It is structured hierarchially so that the user may skip through it to the points of interest to him. The initial information is a general outline of what the system is supposed to be able to do for the user and how to go about using it. This part gives an explanation of the details for using the system, and is similar to the standard introductory material needed for any on-line system.

In addition to the general outline on the use of the system, there is a section of more specific information about the subject matter contained in the modules. Here the user interrogates the system about his specific interest topics and an effort is made to inform him of the content of the modules and to show how he can or cannot be helped. The emphasis here is on subject matter and on what topics are covered by the modules and what are not.

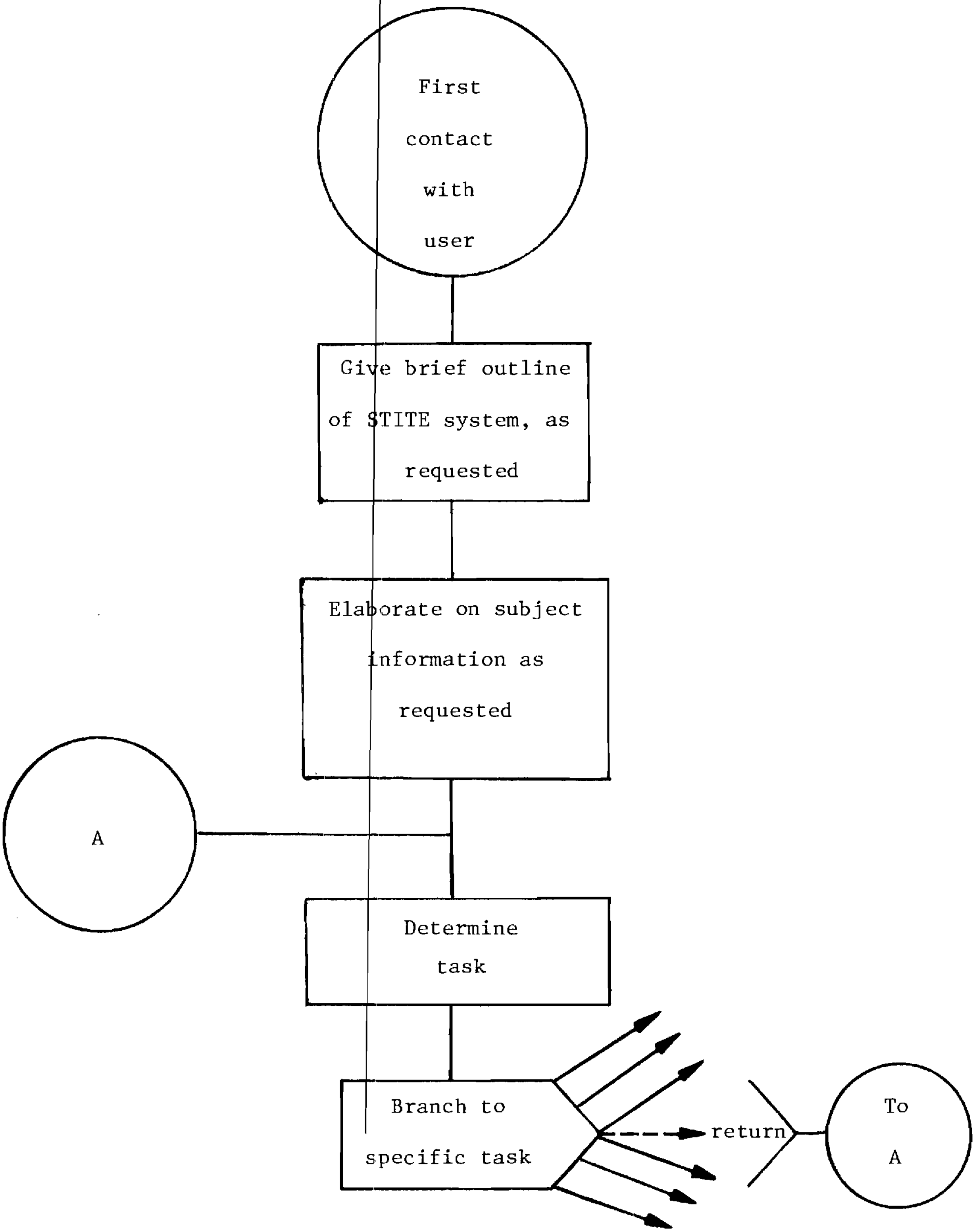
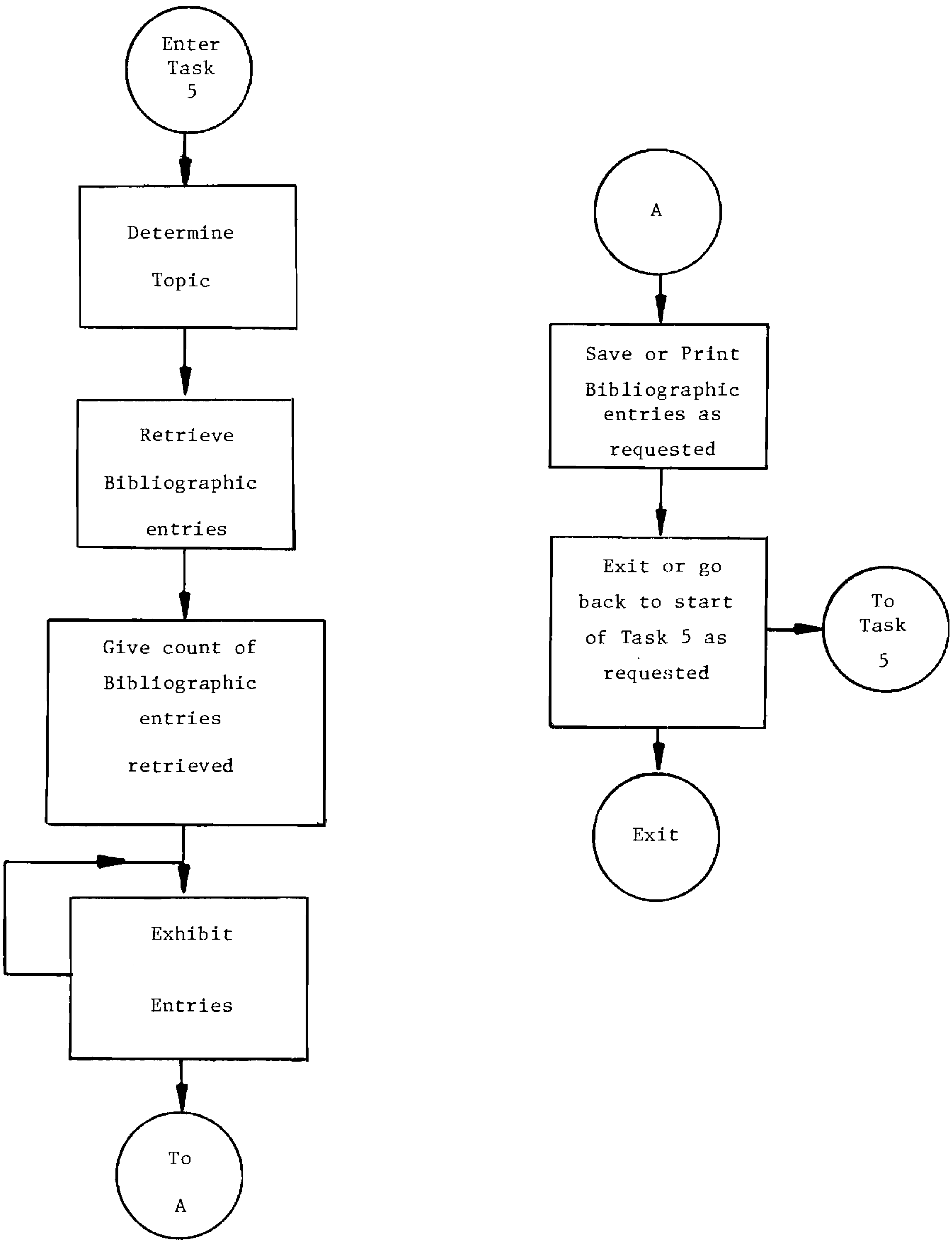

First
contact
with
user
Give brief outline of STITE system, as requested
Elaborate on subject information as requested
A
Determine task
Branch to specific task
return
To A

Fig. 3



Enter
Task
5
Determine
Topic
Retrieve
Bibliographic
entries
Give count of
Bibliographic
entries
retrieved
Exhibit
Entries
To
A
A
Save or Print
Bibliographic
entries as
requested
Exit or go
back to start
of Task 5 as
requested
To
Task
5
Exit

Fig. 4

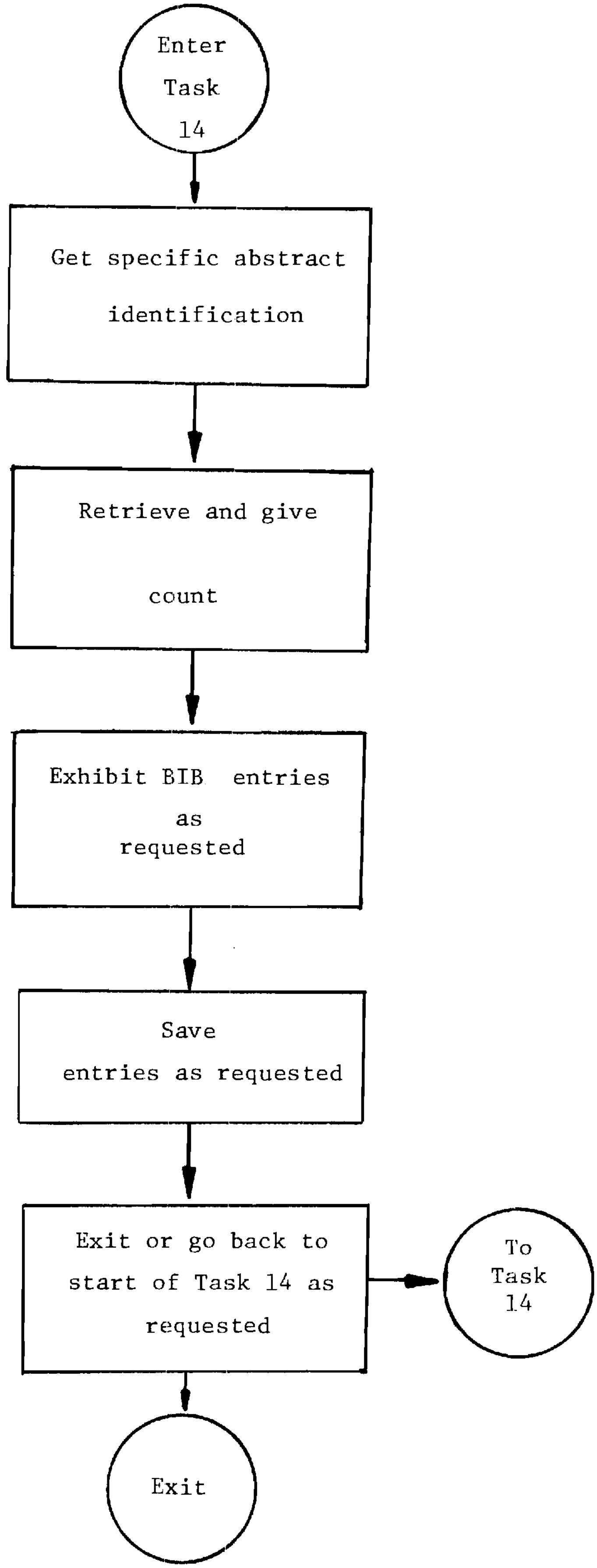

Enter Task 14
Get specific abstract identification
Retrieve and give count
Exhibit BIB entries as requested
Save entries as requested
Exit or go back to start of Task 14 as requested
To Task 14
Exit

Fig. 5

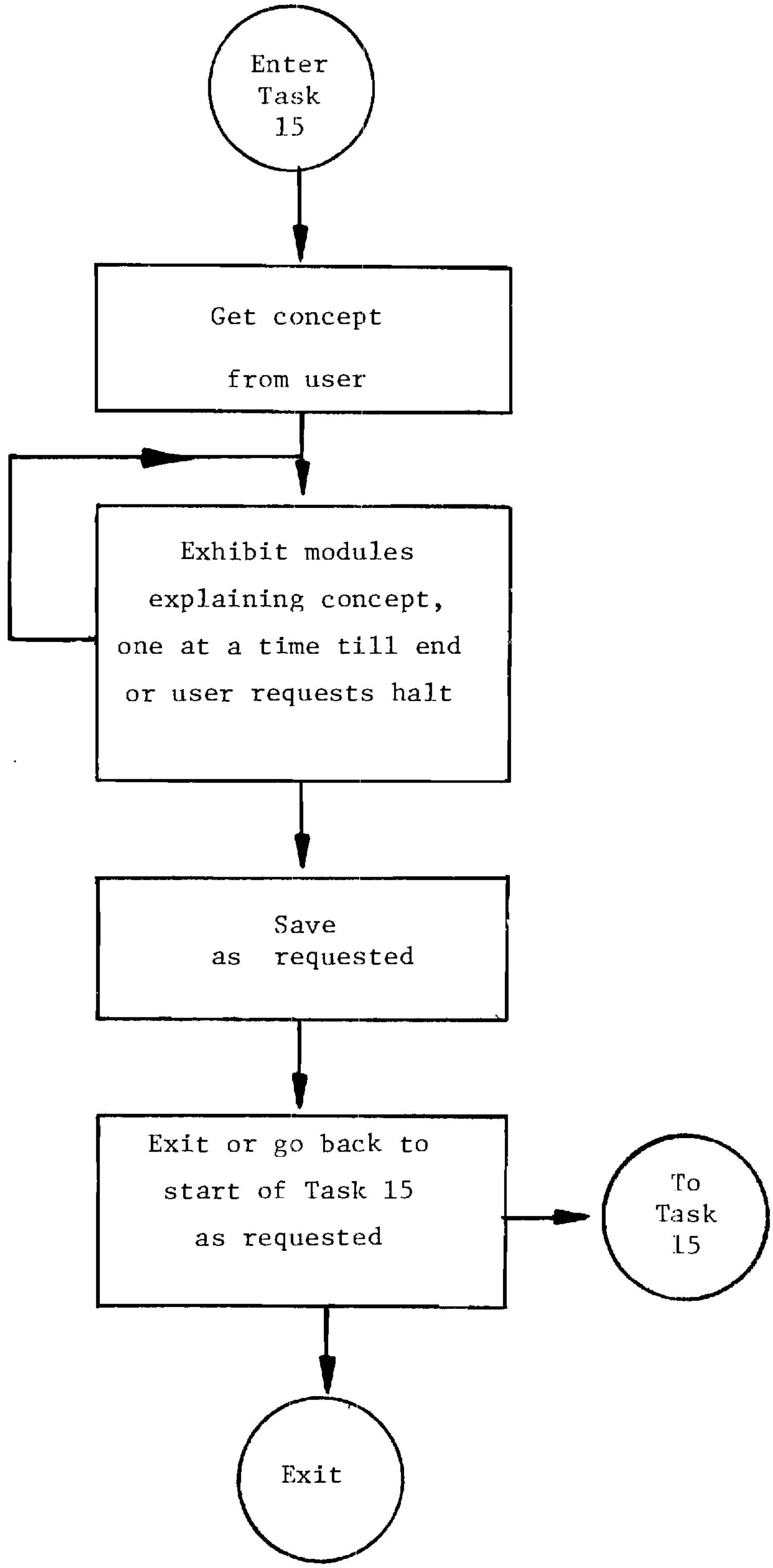

Enter
Task
15
Get concept
from user
Exhibit modules
explaining concept,
one at a time till end
or user requests halt
Save
as requested
Exit or go back to
start of Task 15
as requested
To
Task
15
Exit

Fig. 6

Next there is a brief discussion of the kinds of tasks available. The following tasks from the list of potential tasks described previously in a STITE Progress Report (2) (and in Section IV.1) have been implemented:

Task No. 2 - Preparation of narrative presentatives (relevant theorems only)

Task No. 5 - Compilation of bibliographies on selected documents

Task No. 8 - Retrieval of all abstracts related to a specific abstract

Task No. 14 - Retrieval of all abstracts related to a specific abstract

Task No. 15 - Presentation of concept definitions and explications

Variations on portions of the other major tasks have also been implemented in addition to other related capabilities.

Finally an explanation of the procedure for invoking or calling specific tasks is given.

The entire users instruction dialogue is presented in Appendix A.

At the end of the instruction dialogue, the user arrives at task selection at which point he may call for a specific task implementation. Since these tasks are complex enough, the user may request a printed copy of these task descriptions. If no printed copy is immediately available, entering request for "help" will review for him the list of tasks and their details.

The description of the tasks, what they accomplish and how they are called for is given in the already referred to Appendix A.

Appendix B gives an example or sample dialogue with a typical user with the system operating in user's mode.

IV.3.3.1 Degree or Order of Relationship

One of the more important concepts used in the STITE system is that of relatedness and degree or order of relatedness.

Kinds of relations for the modules are described in some detail in STITE Report #3 and some of these are here briefly reviewed (3):

> The homonymic relation—two modules are said to be homonymicly related if both are about the same main topic (i.e. have a common A field).
>
> The contexual relation—two modules are said to be contexually related if the two have a common topic among the major terms used to discuss or describe the main topic (i.e. have a common term in the B field).
>
> The explicational relation—two modules are said to be explicationally related if they have a common term in the A field of one module and in the B field of the other.

Based on the above relations (and others if desired) one may construct sets of modules connected by these relations. For example, the system contains three modules contexually related by the term 'hamiltonian graph' (i.e. all contain 'hamiltonian graph' in the B field—see Fig. 7a).

Next consider all the additional modules contexually related to this first set. Figure 7b illustrates this. It is clear that this process can be iterated to any order or until the process terminates because some kind of closure is exhibited, i.e. no more contexual relations are found outside of those already included.

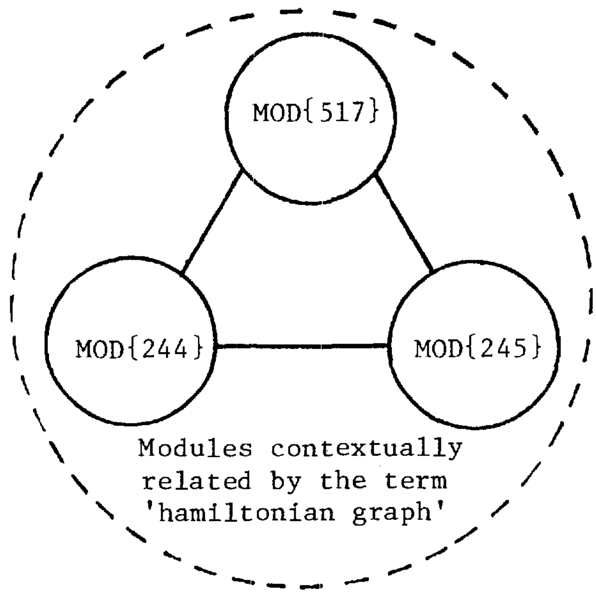
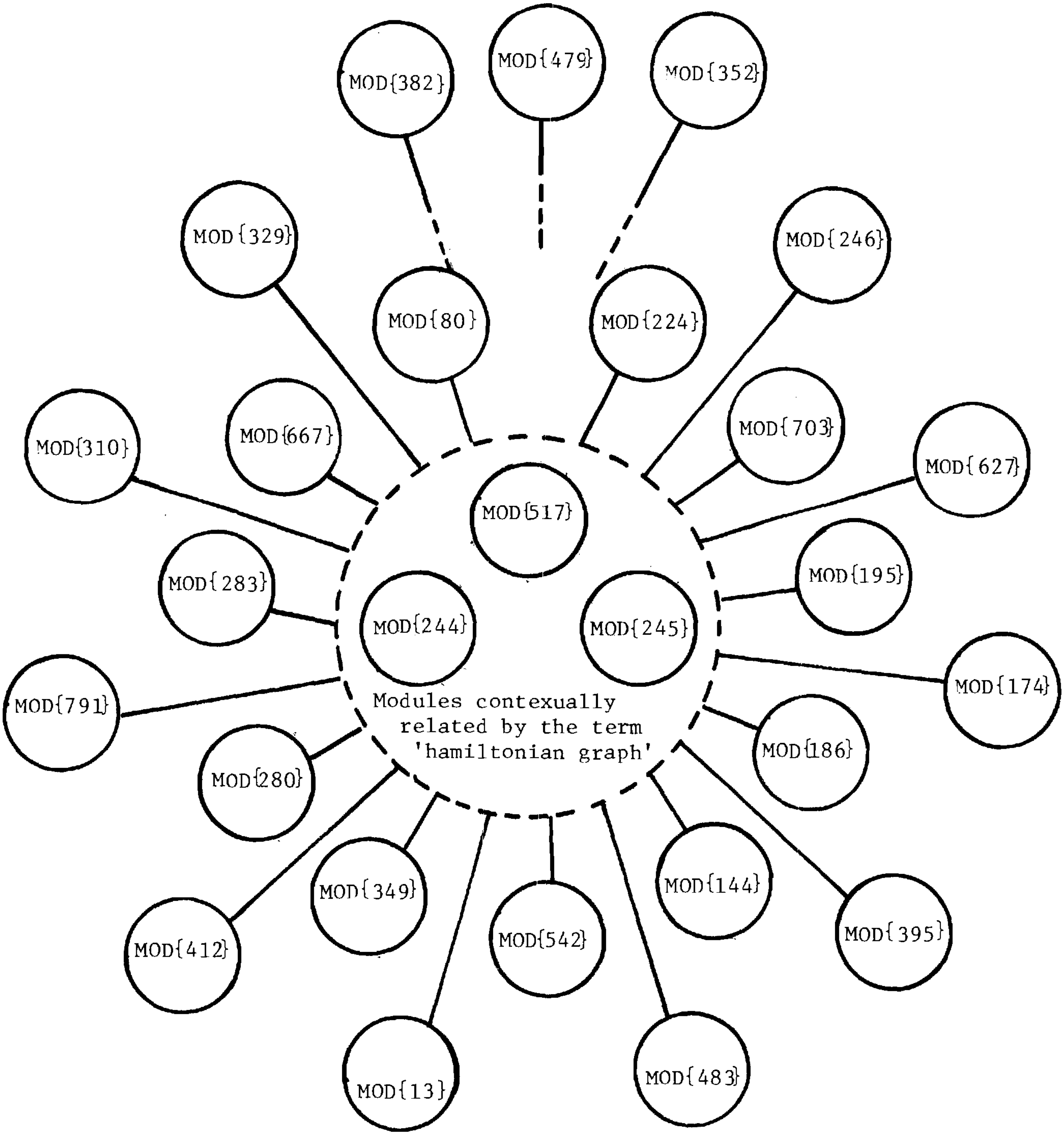

MOD{517}
MOD{244}
MOD{245}
Modules contextually
related by the term
'hamiltonian graph'

Fig. 7a



312 modules (contexually) related to
MOD{517}, MOD{244}, or MOD{245}

Fig. 7b

The process of finding ever-widening sets of related modules can be extended to the other relationships. For example suppose one starts with the (internal) module MOD{29} and finds all modules explicationally related to it. One can now ask for the set of all modules explicationally related to this first set and so on. Figure 8 illustrates this relationship.

It should be clear that there are many modifications of the procedures just described. Most of the tasks available in the user's mode involve variations on the above theme. This includes the ability to make further restrictions on the sets to be retrieved (such as restricting to external modules (bibliographic entries) only, or restricting to definitional modules only (i.e. those of type H{1}) and so forth.

One final explanation is necessary here to describe how the main term (A field) and explanatory terms (B field) were determined for the external modules (bibliographic entries).

To choose the main topic or A field for the bibliographic entries a comparison was made between the key words (both those supplied by the author, and those supplied by the abstracting service) and the title of the article. Key words appearing in titles were then designated A field terms. The key words not appearing in the title were designated as explanatory or B field terms.

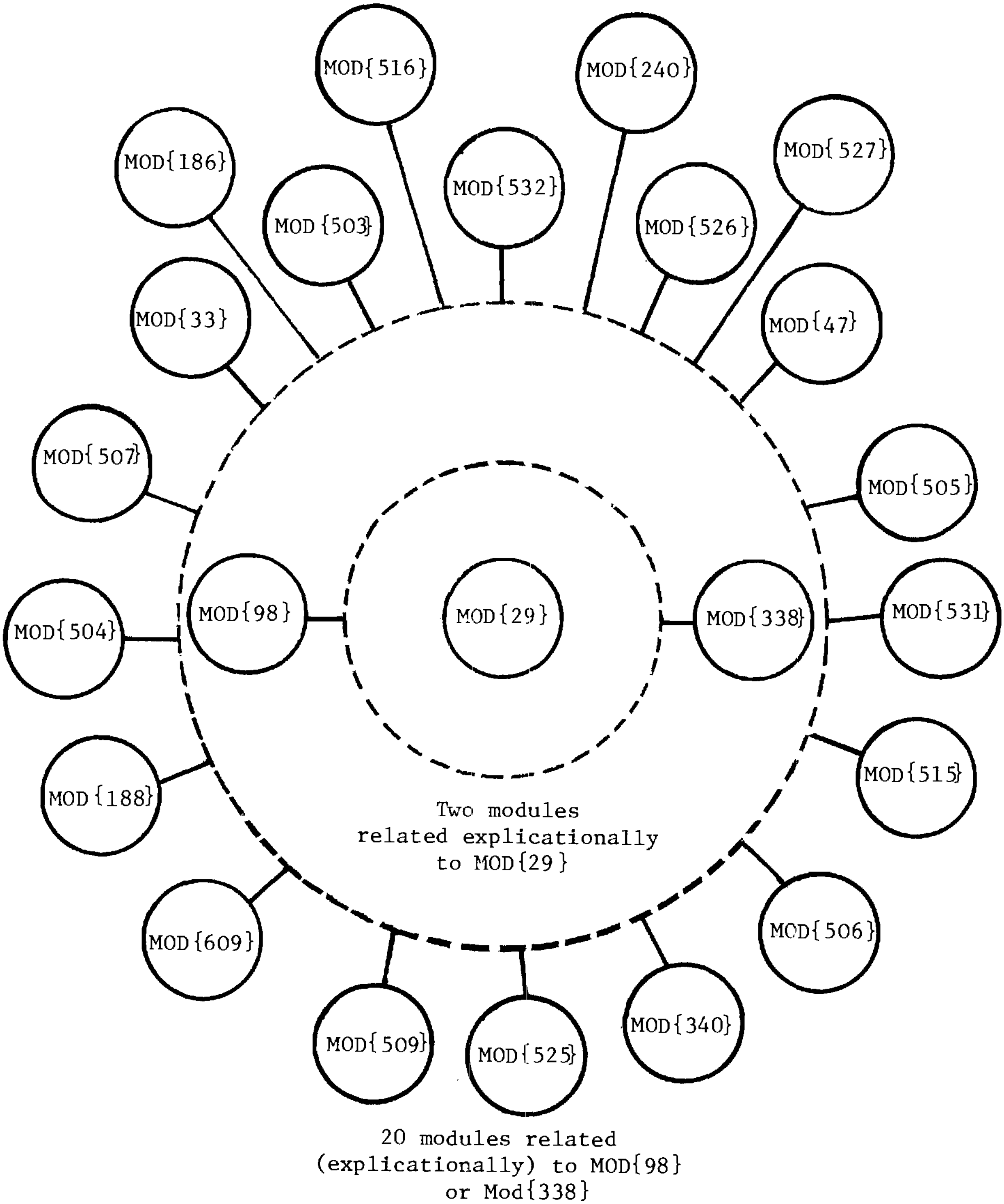

MOD{516}
MOD{240}
MOD{186}
MOD{532}
MOD{527}
MOD{503}
MOD{526}
MOD{33}
MOD{47}
MOD{507}
MOD{505}
MOD{98}
MOD{29}
MOD{338}
MOD{531}
MOD{504}
MOD{188}
MOD{515}
Two modules
related explicationally
to MOD{29}
MOD{609}
MOD{506}
MOD{509}
MOD{525}
MOD{340}
20 modules related
(explicationally) to MOD{98}
or Mod{338}

Fig 8

While this may be considered a rather crude approach to determining main and explanatory topics, it did work reasonably well with the material now available. However, other methods of determining the main topics, such as parsing the sentences in the abstract, should be considered.

To carry out some tasks, especially the open-ended ones, a command language is provided. This is not a conversational language but a set of commands that allows one to manipulate, build, and rearrange the structures to suite the needs. The command language itself is quite simple and is interpreted by the system. This mode is referred to as the Data Base Manager's Mode and is primarily for the experienced user. However, before discussing in detail the Data Base Manager's Mode a description of the file structure will be given.

IV.3.4 The Physical and Logical Structure of the Files

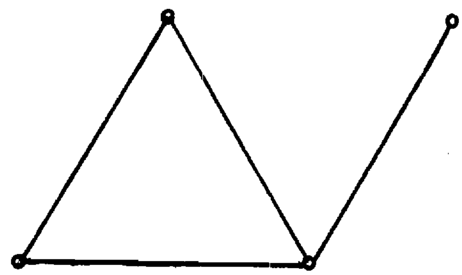
Previous reports have outlined in a general way the organization of the automated STITE system. Some details of the structure of the information files, which will make possible the kinds of action desired, will now be given. Basically the file organization is a relational type.

There are two distinct kinds of records, records of internal information, or modules, and bibliographic data records embodying external information.

A bibliographic data record indicates the author and title of the document, title, date, and page number for a journal article, report number and sponsoring agency for a technical report. Furthermore, it contains an abstract of the document, a list of key words which have been used to index the document, and the title of the index or abstract journal, with its date and abstract number in which the document appears.

The information structure of the internal module has been described in an earlier STITE report (3). Figure 9 illustrates typical internal module.

530



A	B	C WIL/1/29
simple graph	vertex-set	
	edge-set	F 2
	pair	
	edge	H &
	vertex	
	set	
	family	D 1

629

Given any graph G, an edge-sequence in G is finite sequence of edges of the form

$$\left\{v_0, v_1\right\}, \left\{v_1, v_2\right\}, \ldots, \; , \left\{v_{m-1}, v_m\right\}$$

(also denoted by v_0, v_1 ... v_m). It is clear that an edge-sequence has the property that any two consecutive edges are either adjacent or identical; however, an arbitrary sequence of edges of G which has this property is not necessarily an edge-sequence.

A	B	C WIL/1/128
edge-sequence	finite sequence	
	edge	F 1
	form	
	consecutive edges	H 1
	adjacent edges	
	identical edges	D 1
	arbitrary sequence	

Modules 530 and 629

Fig. 9

The fundamental unit of a file is the record which, if not specified, refers to either an internal or external module. The records of both are organized in a similar manner and so will be discussed together.

A record is considered to have a name and an associated set. The record name is an alphanumeric string consisting of a number set off by curly brackets "{}" and prefixed by BIB if it is a bibliographic data record or by MOD if it is an internal module. The associated set consists of a collection of pointers designating fields or attributes of the record.

The associated set of each record consists of a collection of fields Each field has a field name, field value, and associated set. The name and value of a field are joined in a single alphabetic string with the value set off by curly brackets. Field names consist of one, two, or a three letter string. For example, the name and value (attribute) for the A field of module 629 of Figure 6 would be A{edge-sequence}, the B field name and value would be B{finite sequence}, B{edge}, B{form}, etc.

Each field value (except the text) also has an associated set. Sets associated with field values can have such things as the list of all records containing that particular field value (the inversion) or any other relational information deemed relevant or useful. Initially most of these sets are null, but the capability of an association set for a field value will always be present except for the free text field. The free text fields do not have association sets.

The concept of field is quite general so that it is possible to assign new fields to records as needed. Also, in discussing the structure of the files, it will be seen that the distinction between file, record, and field is rather artificial and that all "logical" units are treated in a unified manner.

Physically the files are divided into two kinds: alphabetic information (called α-files) and pointer information. The alphabetic information is stored as alpha-numeric strings with distinct units separated by a special character (low value). The pointer information is stored as a collection of binary (machine language) integers.

Alphabetic files contain all the names, values, and free text information. The pointer information shows the form of the association sets and indicates such things as the structure of a record or file.

The pointer information is of two types: hash references and association sets. These are not separated since the structure of the two is very similar, as well as the frequency and mode of reference.

Figure 10 gives a schematic representation of the file structure and linkages for a part of record Module 629. Details on the hash file and the structure of the association sets are given in STITE Report #4 (4).

IV.3.5 The Data Base Manager's Mode and Language

For carrying out certain tasks and for performing other desired operations, a command language for manipulating the records and creating structures has been implemented.

Looking at the records or structures being manipulated, one notes that they are basically names and sets of names. Associated with each name is a set called its associated set. An associated set is a true set (in the mathematical sense) of names (since the order is not relevant and no duplicates will be allowed). Some of the association sets are empty. Set theoretical operations are defined and implemented on these association sets.

The commands are in a very simple form and restricted to unary and binary operations. One of the operands is designated by a P or pointing register; the other, if there is one, is named in the command. (This is reminiscent of a single address machine language with an "accumulator" register, except that here the "accumulator" register does not hold the item to be operated on but only points to it.) The item pointed to by the P or pointing register will sometimes be referred to as just P. Thus, the command, <u>point X</u>, points P to X, that is, sets the P or pointer register pointing to X. The command <u>attach Y</u> adds Y

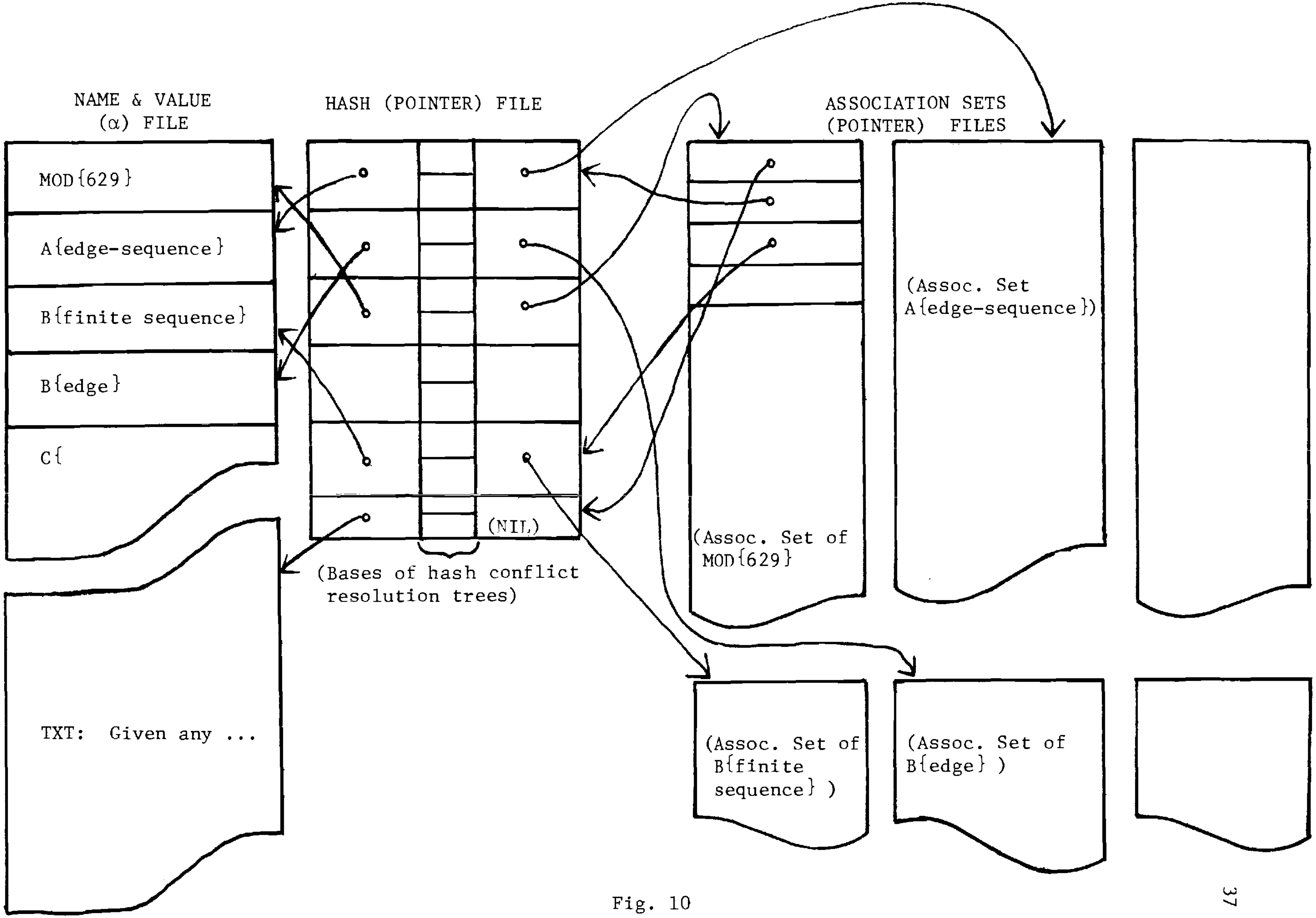


Fig. 10
Example of File Structure
(For "Module 629" of Fig. 9)

to the P set, that is, adds the name Y to the association set of the name pointed to by the P or pointer register.

This command language is outlined in Appendix C.

Some examples of the use of these commands to perform some elementary operations follows:

1) Create Q so that it is the union of X and Y. That is, the association set of Q is the union of the association sets of X and Y.

```
create Q
add list X
add list Y
```

2) Remove all items from the association set of R, i.e., make R the null set:

```
point R
delete set R
```

or

```
create Z
point R
delete setnot Z
```

3) Display those items on the association set of X but not on the association set of Y:

```
create TEMP
add set X
delete set Y
display set
```

4) See if X is on the union of Y and Z:

```
create T2
add set Y
add set Z
is member X
```

Following each command is a response by the system to the user giving an indication that the command has been carried out or cannot be obeyed. Examples of commands that cannot or will not be carried out are as follows:

1)	create X	Not allowed if the name X already exists as a recognizable name
2)	point X	Not allowed if the X does not already exist as a recognizable name
3)	remove X attach X add set X etc.	Not allowed if the name X does not already exist as a recognizable name, or if P register has not yet been set
4)	display name display set count etc.	Not allowed if pointer or P register has not yet been set to point to a recognizable name

While this is a very primitive type command structure for manipulation of the association sets, it allows the implementation of the desired processes. Such a simple scheme was chosen for ease and speed of implementation without the need for an elaborate language implementation study which is not directly among the goals of this project. Appendix C gives the (optional) instructions displayed at the time the DBM Mode is entered (or in response to the "help" request). Appendix D gives an example dialogue using the Data Base Manager's Mode with examples of some retrievals.

IV.3.6 Some Characteristics of Programs and Paging System

The computer programs for processing the records are written in the C language (5) for the PDP 11 computer. C is an Algol-like language particularly well adapted to the PDP 11 computer and operates under the Unix operating system (6), a multiuser system for the PDP 11.

The PDP 11 configuration in use has 104 K bytes of core and about 200 M bytes of disk storage, although, of course, not all of this is available to a single user. The maximum core allowed a single user by the Unix system is about 56 K bytes. The example system is using about 2 M bytes of disk storage.

Every effort was made in the programming to use modular and hierarchical techniques. The C language is reasonably well suited to these methods.

Because of the limited core the first requirement of the programs is for a (data) overlay or paging system, and this is incorporated at the lowest level. Since it is the linkages that are expected to be referenced most frequently, these were incorporated into the paging system. The alphabetic information is needed mainly for communication with the user and referenced as needed, and is not involved in the "automatic" paging.

Many of the characteristics of the link-paging system are determined by the machine being used (PDP 11) and the operating system (Unix). The fixed page size of 512 bytes was chosen to fit the hardware/software. A link consists of four addresses, each address being 3 bytes, 2 bytes to indicate the page number and one for the links within the page. This allows the spanning of 64 K pages of linkage data, the maximum allowed to a user (by Unix). This allows 42 links per page with 9 bytes left over for some housekeeping chores.

A system of N buffers is set up, with each buffer holding one page. N is a program variable and is generally between 32 and 64, consistent with other core requirements. The linkage pages are moved in and out of the buffer areas as needed with a directory kept of which are in core. In addition, the directory keeps a record of the manner in which the pages are referenced. Those pages referenced least recently and/or least frequently are first to be moved out when a new page is required to be moved in.

The concept here is that during a given time interval in the running of the program, certain pages will be referenced with high frequency and other less often. The particular set of pages so referenced will change with time or with the demands of the program. The paging system is expected to adapt to these changes and to adjust in a statistical manner to the changing page requirements. It is also expected that the linkages will cluster so that there is a high probability that links will point to others on the same page. While no systematic efforts to produce clustering are made, this particular application and the manner in which the links are initially assigned encourage clustering. However, it is recognized that if extensive changing and reassignment of the linkages occurs, the clustering or correlation of the links within a page will disappear and become random or incoherent. This could be corrected by more sophisticated garbage collection and link assignment algorithms. However rough experiments showed that the more elaborate assignment methods generated enough overhead to cancel the anticipated gain. Details of the paging algorithms are given in the fourth STITE Progress Report (4).

IV.3.7 Structure of the Programs

The C language (5) is a procedure or subroutine oriented language and suited to modular and hierarchical programming techniques. Here will be given some of the structural characteristics of the programs.

The basic building unit in C is the procedure. A procedure is a more or less closed set of code that is called or invoked as a unit. It can have within it calls on other procedures (including itself). Procedures communicate with each other both through parameters and global variables. (While it is considered that procedure communication through global variables is "dangerous", i.e., an error-prone method of doing things, this would be most difficult and awkward to avoid in the C language. The best that can be done is to keep global variables to a minimum.)

The procedures are grouped into sets (called blocks) that perform particular duties. Sets of these blocks are then built into programs that perform higher order tasks. Sets of programs can also be grouped together to perform yet higher order tasks. The program set is the top of the hierarchy in the C-Unix system on the PDP 11.

Figure 11 shows blocked diagrams of STITE system programs including the Data Base Manager's program, and the user's system and the (internal) modules processing program in terms of the program modules; only the top most block is different in all these programs.

Each block represents a collection of procedures used to perform the indicated tasks. Global variables are global only within a block and may not be referenced outside of that block. Procedures within a block may reference each other or those in a lower block. Thus, for example, the set manipulation procedures reference each other and those of link-paging and α-file management, but link-paging and α-file management never reference each other nor the global variables associated with each others' blocks. This hierarchy structure makes debugging and trouble shooting easier since the blocks can be checked out in order from bottom to top (or top to bottom).

Appendix E lists these programs.

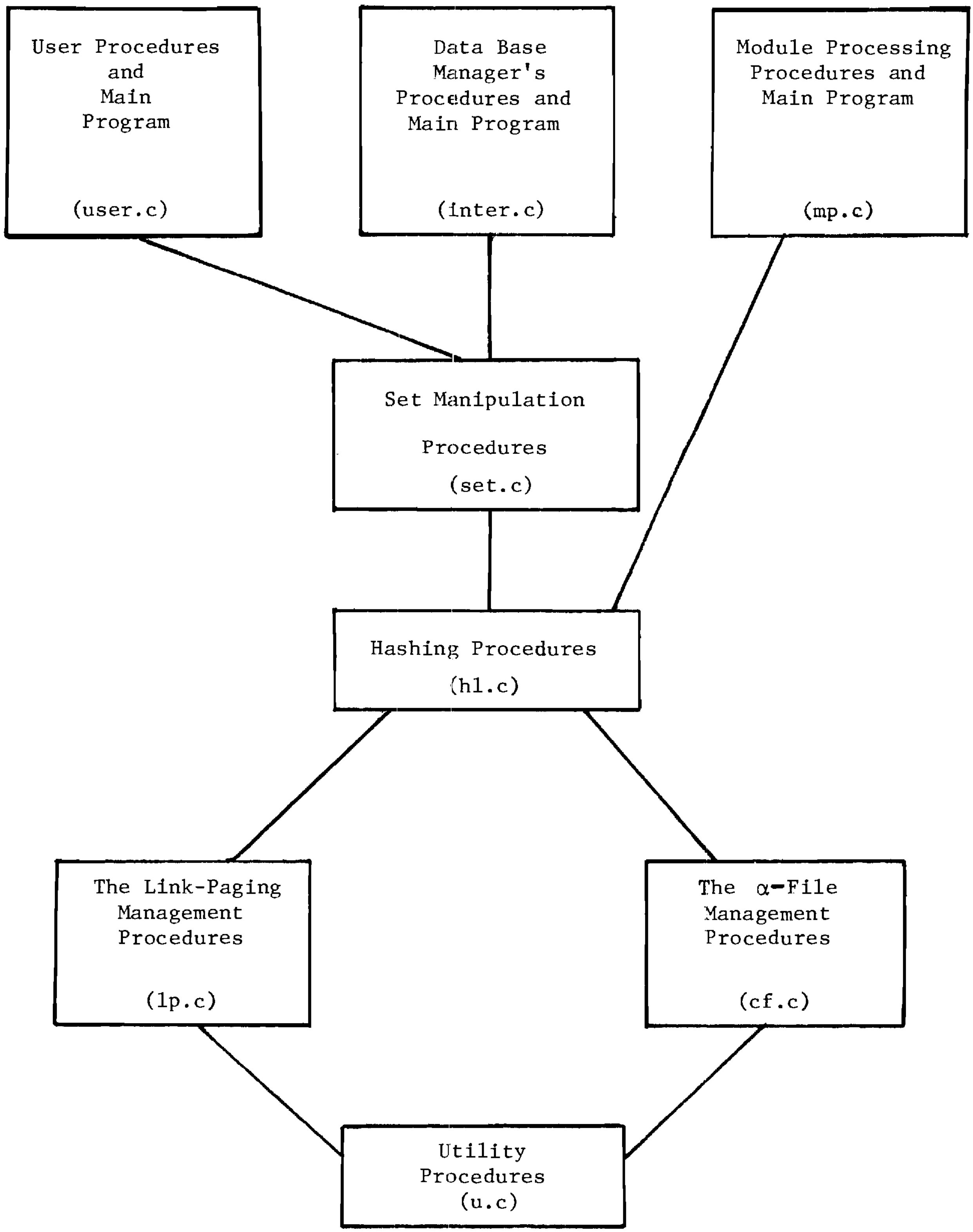

User Procedures
and
Main
Program
(user.c)
Data Base
Manager's
Procedures and
Main Program
(inter.c)
Module Processing
Procedures and
Main Program
(mp.c)
Set Manipulation
Procedures
(set.c)
Hashing Procedures
(hl.c)
The Link-Paging
Management
Procedures
(lp.c)
The α-File
Management
Procedures
(cf.c)
Utility
Procedures
(u.c)

Fig. 11

V. AVENUES OF FURTHER DEVELOPMENT OF THE STITE-SYSTEM

Various aspects of transfer of scientific and technical information to educational and learning systems, which have been studied, and the design of an experimental STITE system, which was one of the main accomplishments of this project, opened numerous new avenues for follow-up investigations and developments. The most important of these are:

(1) User evaluation of the STITE system under realistic operating conditions. Retaining the present data base on graph theory and applications, arrangements would have to be made with a selected group of educators to use the system for actual educational purposes in teaching graph theory courses or seminars. To make this exercise meaningful, one would need to connect the STITE system on-line to some actual information retrieval system, since otherwise the obvious limitations of the system would make such an actual utilization hardly attractive.

(2) Evaluation of performance of the STITE system on a selected spectrum of data bases. The "internal knowledge" of the experimental system is limited to the subject matter of graph theory, whereby the selection of the subject matter was somewhat arbitrary. One would like to explore how effectively the artificial intelligence component of the system (i.e. internal modules of the system and their syntactic and semantic structures) could be constructed for different subject areas and how well the system would perform on these different data bases.

(3) Investigation of optimal trade-offs between the economy of performance of the STITE System in various tasks and the size of the "internal knowledge", i.e. the amount of structured information stored in the system in the form of internal modules etc. Intuitively, the larger in size is this component of the system, the more time will be required to process information for various tasks. For all practical purposes, methods need to be developed for finding an optimal balance between the volume of "internal knowledge" and economy of performance.

(4) Development of add-on components of the system which would improve the interaction with the user and the performance for various tasks. Two such subsystems would be of particular interest: a subsystem for interactive and automatic generation of user interest profiles and subsystems for preselecting external information for retrieval purposes, i.e. preselecting subsets of external information for detailed search and high recall retrieval. Some clustering programs have been already developed by us with this purpose in mind.

These are just the major follow-up tasks which seem to be worthwhile pursuing on the road to a fully operational STITE System. It is by no means an exhaustive list of all possible extensions.

VI. REFERENCES

1. Zunde, Pranas (Project Director). Scientific and Technical Information Transfer for Education (STITE). Atlanta, Georgia, School of Information and Computer Science, Georgia Institute of Technology, July 1973. Progress Report, NSF Grant No. GN-36114.

2. Zunde, Pranas (Project Director). Scientific and Technical Information Transfer for Education (STITE). Atlanta, Georgia, School of Information and Computer Science, Georgia Institute of Technology, December 1973. Progress Report No. 2, NSF Grant No. GN-36114.

3. Zunde, Pranas (Project Director). Scientific and Technical Information Transfer for Education (STITE). Atlanta, Georgia, School of Information and Computer Science, Georgia Institute of Technology, June, 1974. Progress Report No. 3, NSF Grant No. GN-36114.

4. Zunde, Pranas (Project Director). Scientific and Technical Information Transfer for Education (STITE). Atlanta, Georgia, School of Information and Computer Science, Georgia Institute of Technology, June, 1975. Progress Report No. 4, NSF Grant No. GN-36114.

5. Bohnert, H., and Kochen, M. "The Automated Multilevel Encyclopedia as a New Mode of Scientific Communication." Proceedings of ADI, 2:269-270 (1963).

6. Brady, E.L., and Branscomb, L.M. "Information for a Changing Society." Science, 1975:961-966 (March 3, 1972).

7. Brown, G.W., et al. EDUNET--Report of the Summer Study on Information Networks. New York, Wiley, 1967.

8. Kochen, M. "Information Science and Computer Aids in Education." in Schultz, L., ed., The Information Bazaar, Philadelphia, Pa., the College of Physicians, 1969, pp. 209-226.

9. Badre, A.N., Hughes, D.S., Ting, T.C., and Zunde, P. Science information transfer for learning. P. Zunde (ed.), Information Utilities. Proceedings of the ASIS Annual Meeting, Vol. 11, ASIS, Washington, D.C., 1974, pp. 120-124.

10. Ritchie, Dennis M., C Reference Manual, Bell Telephone Laboratories, Murray Hill, N.J. 07974.

11. Thompson, K., and D.M. Ritchie. Unix Programmer's Manual. Sixth Edition. Bell Telephone Laboratories, Inc., May, 1975.

APPENDIX A

The User's Instruction Dialogue for the STITE Data Base Management System

The major functions of the STITE system were broken down into tasks and discussed in the STITE Report#2 (Dec 1973). Seventeen tasks were identified and analyzed. All of these centered around the use of the "internal" information, structured as modules and described in STITE Report #3.

Data base elements are records of two kinds: modules of internal information (called moditems) and bibliographic records (called bibitems). The modules are described in some detail in SR # 3. Each consists of textual material and additional information about the textual material. This information associated with the text is arranged in fields, with the principal topic being designated as the A field. B field contains a set of terms which are used to describe or explain the principal topic, or A field. C field indicates the source document from which the textual material is taken, F field specifies the form of the text (ie, English language, chart, etc.), H field indicates type of informatin (ie, definition, theorem, or problem), and D field refers to the level of difficulty of the textual material.

The most significant feature of the Stite system is the ability to browse through the bibliographic material (and modules) in an associative manner. That is, relations of various types between groups of bibliographic entries can be examined.

The principal types of relationships that are considered significant are

a) The homonymic relation: two bibliographic or module items are said to be homonymically related if they are about the same main topic (ie have the same A field term).

b) The explication relation: two bibliographic or module items are said to be explicationally related if one is an explination or explication of the other (ie the same term appears in the B field of one and the A field of the other).

c) The contexual relation: two items are said to be contexually related if both use a significant common term to describe, define or otherwise demonstrate their main topic (ie have a term common their B fields).

This is considered the distinctive feature of the Stite system: to be able to find collections of items (bibliograph or module) that are meaningfully related to one another. Chains of these relationships can also be examined. That is, a connected path or tree of bibliographic or module relationships can be created using either the explicaticanal or the contexual relation. These chains start with a given term or topic and find a sequence of module of bibliographic entries that connect to a second given term, where the connection can be either contexual or explicational. The entries along the chain or tree can then be displayed.

Also certain specific tasks, outlined and discussed in STITE Report #2, have been implemented more or less as originally described. These include:

Task No. 2. Preparation of Narrative Presentations. (Only the part about presentitng sets of theorems relevant to a topic or sub-topic has been implemented.)

Input: Topics or sub-topics of interest.

Output: Set of relevant theorems (with proofs if available).

Task No. 4. Presentation of Problems Under Specific Topics or sub-topics.

Input: Topic, sub-topic, or key word.

Output: A set of problems relevant to the given key word.

Task No. 5 (and No. 8). Presentation of Bibliography Under a Topic or Sub-topic.
Input: Subjects, terms, topics, sub-topics or concepts.
Output: A set of references under headings of each topic or subtopic.

Task No. 14. Retrieval of Abstracts Related to a Specific Abstract.
Input: The Reference number of the abstract.
Output: A sequence of related abstracts or bibliographic entries.

Task No. 15. Presentation of Definitions of Concepts.
Input: Terms, topics, sub-topics, or concepts.
Output: The relevant definitions.

Finally, unions, intersections, and differences of sets of modules and bibliographic entries can be construced and manipulated.

The Stitc Users system is a yes-no driven, agenda type dialog system. Most requests for a user response are to be answered 'yes' or no (or just y' or n). When other type input is requested it should be clear from the question what is wanted. Input is always free field, never more than one line at a time, and terminated by a carriage return.

At various stages the system will attempt to inform you in some detail about what it is doing and what you will need to do. However these instuction monologues are broken every 15 lines or so with a question giving you the option of continuing the discussion or skipping into the next material on the agenda. While you may skip forward in the agenda this way, you may not back up. One can however terminate the program and start over. (To terminate hit the "rub out" key in response to any question or request for input; the "%" indicates you have returned to the UNIX operating system.)

The sequencing or agenda is as follows:

1. Introduction and preliminary instructions (we are in that now).
2. Discussion of the topics or subject matter in the system. (Here you may look at some of the subject modules if you choose.)
3. Discussion of the tasks presently implemented.
4. A choice of tasks is made by the user.
5. Cycle through the options associated with the choosen task. Options include looking at or saving material retrieved and extending (or narrowing?) range and type of retrieval.

The user should also be aware that there is another manner of accessing this data base which allows cosiderably more freedom and detailed manipulation of the elements. In effect it allows the user to become his own data base manager; in doing this however one assumes considerably more responsibility for the integrity of the data base and the manipulation process is more complex than the users mode. To enter this "data base manager s" mode, exit from the users mode and give inter.a .

Subject Matter and Topics

The present system deals almost exclusively with the subject matter Graph Theory and its applications. Thus topics within this area will be recognized. Typically for example the topics

simple graph	eulerian graph	digraph
tree	vertex	mapping
edge	chain	circuit

and so forth are represented and discussed. However there are sereral thousand terms or topics represented and discussed in the modules and many more in the bibliographic material (bibitems) that one may ask about. Most of these are only distantly related to graph theory as such and many seemingly unrelated, even remotely.

One may enquire about any topics or terms one chooses and if there are modules or bibitems referring to the topic one may request to see them.

At present the choice of tasks is as follows:

Compilation of bibliographies on related topics (STITE Report #2, Dec 1973 pages 5 & 7, referred to there as Tasks No. 5 and 8).

Compilation of module collections on related topics. (This includes the construction of outlines by topic or subtopic.)

Retrieval of abstracts related to a specific abstract (abstract = bibliographic entry = bibitem; STITE Report #2, Dec 1973 page 10, referred to there as Task NO. 14).

Retrieval of modules associated with a particular module.

Retrieval of intersections (or unions or differences) of collections of modules and/or bibliographic entries that have common terms in various fields.

Creation of connections or paths between related terms or topics. (Terms are related through modules or bibliographic entries in the sense that an entry contains both terms (in either the A or B field).)

Retrieval of theorems relevant to a particular topic (part of Task No. 2 in STITE Report #2, page 4).

Presentation of problems relevant to a prticular topic (referred to as Task No. 4 in SR #2, p. 5).

Presentation of definitions of concepts (Task No. 15 in SR #2, p. 10).

Compilation of Bibiliographic Entries (bibitems) on Related Topics:

This task can be carried out in three different formats--

Format AB) by the ABiteration (see SR #4 page 60); successive orders of relatedness are determined by--

1) finding the union of all module entries (moditems) and bibitems for the topic(s) as A field [all items homonymicly related through this topic(s)],

2) then finding all the B fields of these items,

3) converting all these B fields to A fields (with the same values),

4) finding all bibitems for these A fields [ie, all items explicationally related to the first set].

This sequence can then be repeated as often as the user chooses.

Format BA) by the BAiteration; same as the ABiteration of Format a) with B and A interchanged [ie, first all contexually related items, then those related (reverse) explicationally to the first set of items].

Format BB) by the BBiteration; same as the ABiteration except that one starts

```
        with B fields and ends with B fields [uses only the contexual
        relationship].

To do a compilation on selected topics(related by bibitems) via
        Format AB) request "1",
        Format BA) request "3",
        Format BB) request "5".
Starting topic(s) will be requested.

To do a compilation on selected topics(related by moditems) via
        Format AB) request "2",
        Format BA) request "4",
        Format. BB) request "6".
Starting topic(s) will be requested.

Requests such as "2&1", "1&2", "4&3", "5&6", etc.. will give BOTH
the modules and the bibliographic items for the corresponding request.

Retrieval of Abstracts Related to a Specific Abstract:
        This task can be carried out in three different formats--
Format AB) by the ABiteration (see SR #4 page 60); successive orders of
        relatedness are determined by--
        1) finding the union of all A field topics for the given bibitem(s),

        [the homonimic relation]
        2) taking the union of the moditems and bibitems for these topics as
        A field,
        3) then finding all the B fields of these items,
        4) converting all the B fields to A fields (with the same values),
        5) finding all bibitems for these A fields [the explicational relation].
        This sequence can be repeated as often as the user chooses.
Format BA) by BAiteration; same as the ABiteration of Format a) with
        B and A interchanged [contexual followed by (reverse) explicational].
Format BB) by BBiteration; same as the ABiteration except that one starts
        with B fields and ends with B fields [the contexual relationaship].

To do a compilation on selected bibitems via
        Format AB) request "7",
        Format BA) request "9",
        Format BB) request "11".
Starting bibitem number(s) will be requested.

To do a compilation on selected moditems via
        Format AB) request "8",
        Format BA) request "10",
        Format BB) request "12".
Starting moditem number(s) will be requested.

Retrieve intersections (or unions or differences) of collections of modules
and/or bibitems:
        This task can be carried out in several different ways--
Format MODS) does an intersection(union, difference) of modules only on
        given fields and values.
Format BIBS) does an intersection(union, difference) of bibitems only on
        given fields and values.
Format MODS-BIBS) does intersection (union, difference) of both modules and
```

```
        bibitems on the given field and values.
In each case the resulting collection can not only be
retrieved but saved and named for further reference.

To generate intersections (or unions or differences) of collections of modules
and/or bibitems via
        Format MOD) request "18",
        Format BIB) request "17",
        Format BIB-MOD) request "18&17" or "17&18".

Creation of connections or paths amoung terms:
        This task creates chains of connections or associations
        of terms (connected through the modules or bibitems).
        There  are three possible formats here--
Format AB) the connecction is through the first term appearing as
        an A term and collecting all possible B terms;  the next
        connection is performed by converting all these B terms to
        A terms, and so iterating along the paths until the appropriate
        final term is found. [The mod and bib items on the paths are
        (reverse) explicationally related.]
Format BA) same as the AB format with A and B interchanged.
        [The items are explicationally related.]
Format BB) The connections are through all terms appearing as
        B terms;  the paths result from all possible connections
        to the final term. [The iterms are contexually related by the tems.]

To generate connections or paths amoung topics or subtopics via
        Format AB) request "13",
        Format BA) request "14",
        Format BB) request "15".
Starting and ending terms will be requested.

To  perform the implemented versions  of the Task No. 2, 4, 5, 14, or
15 (discussed in SR #2):

        For Task No.  2;  request "T-2",
        For Task No.  4;  request "T-4",
        For Task No.  5;  request "T-5",
        For Task No. 14;  request "T-14",
        For Task No. 15;  request "T-15".

However there are a variety of ways in which the indicated tasks
can be carried out.  Ie, contexual, homonymic, or explicational
relations (or even reverse explication) can be used to describe
how terms, bibliographic entries, definitions, theorems, etc are
"related" to each other.  These options are included in choosing the
task.  That is:
        For Task No. 5 with contexual relations, request          "T-5-con",
        For Task No. 5 with explicational relations, request     "T-5-exp",
        For Task No. 5 with reverse explication, request         "T-5-rex".
The default option, that is "T-5", gives explication:  "T-5" = "T-5-exp".

The options for Task No. 14 are the same:
        For contexual relations request            "T-14-con",
        For explicational relations request        "T-14-exp",
        For reverse explication request            "T-14-rex",
```

Default is explication; ie, "T-14" = "T-14-exp".

For Tasks No. 2, 4 and 15 the relation options are contexual or homonymic. Thus:

For Task No. 2 with homonymic relation, request "T-2-hom",
For Task No. 2 with contexual relation, request "T-2-con",

and the default is homonymic ("T-2" = "T-2-hom"). And so for Tasks 14 & 15:

"T-4-hom" gives Task No. 4 with homonymic relation,
"T-4-con" gives Task No. 4 with contexual relation,
"T-15-hom" gives Task No. 15 with homonymic relation,
"T-15-con" gives Task No. 15 with contexual relation,

and "T-4" = "T-4-hom", "T-15" = "T-15-hom".

Finally, the task request "help" will cause task choices to be displayed; the task "halt" or just "0" terminates the program.

For carrying out certain tasks and for performing other desired operations, a command language for manipulating the records and creating structures is available.

Looking at the records or structures being manipulated, one notes that they are basiclly names and sets of names. Associated with each name is a set called its associated set. An associated set is just a set in that the order is not relevant and no duplicates are be allowed. Some of the associated sets are empty. Set theoretical operations are defined and implemented on these associated sets.

The commands are in a very simple form and restricted to unary and binary opperations. One of the operands is designated by a P or pointing register; the other, if there is one, is named in the command. (This is reminiscent of a single address machine language with an "accumulator" register, except that the "accumulator" register does not hold the item to be operated on but only points to it.) The item pointed to by the P or pointing register will sometimes be referred to as just P. Thus, the command, point X, puts P to X, that is, puts the P or pointer register pointing to X. The command attach Y adds Y to the P set, that is, adds the name Y to the associated set of the name pointed to by the P or pointer register.

The operations available are:

create x
    Create the name "x", associate the null set, set P register pointing to x
point x
    Set P register to x
attach x
    Put x on the associated set of whatever is pointed by P register
remove x
    Take x off the associated set of whatever is pointed by P
display name
    Display name pointed by P
display set
    Display the associated set of whatever is pointed by P
display secondary
    Display the sets of the set of P
display n-ary N.m
    Displays the sets of the sets of the . . (to a depth N) . .

```
          of P;  if m = 1 only the first element on the Nth set
          will be displayed, if m = 0 all items on the Nth set
          will be displayed
add set x
          Add the items on the x associated set to the items on the set
          pointed by P, eliminating duplicates (the union or OR operation)
delete set x
          Delete from the P set all items on the pointed or P set (the AND NOT
          operation)
delete setnot x
          Delete from the set pointed by P all items not on the x set (the
          intersection or AND operation)
count
          Count the number of items on the pointed of P set, 0 if null
is member x
          Responds yes if x is on the pointed (P) set, no otherwise
relation set x
          Gives set relation between the pointed set and the associated set
          of x.  Response:
                    "identical" if x & P sets identical
                    "superset" if x is a proper superset of P
                    "subset" if x is a proper subset of P
                    (count) of intersection of x & P
                    "disjoint" if x & P sets disjoint
make set null
          Sets the P set to null
on secondary put x
          x is placed on all the sets of the items on the set of P
union secondary x
          The set of P is replaced by a set that is the union of the sets of
          the items on the x set
intersect secondary x
          The set of P is replaced by a set that is the intersection of
          the sets of items on x
exchange x
          Exchanges the sets of x and P;  x then has P's old set and
          P has the set of x
eliminate field F
          All the elements of the P set (that are of the type of the field
          F) are eliminated or removed;  F must be of the form A{ , B{, etc..
          or TXT:
eliminate fieldnot F
          All elements of the P set (that are not of the type of the field F)
          are removed from P;  F must be of the form A{, B{, etc.. or TXT:
ABiterate
          The set of P must consist of field name values (ie of the
          form B{ ... }).  The set of P is replaced as follows:
          1) convert all these B fields to A fields (with the same
          name values), 2) find all the modules for these A fields, 3)
          find all the B fields of all these modules.  The final P set is the
          set of all the B field name values obtained at step 3.
BAiterate
          Same as ABiterate except that the A and B fields are interchanged
BBiterate
          Same as ABiterate except that one starts with B fields (and ends with
          B fields)
```

```
ABconnect to x
        Performs ABiterate until a match on x is obtained;  after each
        iteration the number of items on the P set is given and the
        option of continuing or stopping is given
BAconnect to x
        Same as ABconnect except with A and B fields interchanged
BBconnect to x
        Same as ABconnect except that only B fields are used
ABpath to x
        The set of P must consist of only A field name values
        (ie of the form A{ ... }).  A search is made of the module sequences,
        using ABconnect, until a connection is made with x.  Before starting a
        request for the maximun depth of the search is made.  All paths for
        which a connection is found are printed with the depth of the
        connection specified.
BApath to x
        Same as ABpath with A and B fields interchanged
BBpath to x
        Same as ABpath except that only B fields are used
Rexp x
        All of the elements of the set of x are compared in pairs.  If two
        elements of x have common topic (name value) on their sets that is in
        the A field of one and in the B field of the other,  the two elements
        are said to be explicationally related and the (ordered) pair are put
        on the set of P with the member having the A field match first.
        (SR # 3, page 48).
Rhom x
        All elements of the set of x are compared in pairs.  If two elements
        of x have the common topic in an A field of their sets, the two
        elements of x are said to be homonomicly related and the (ordered) pair
        are put on the set of P.  (SR # 3, page 51).
Rcon x
        Same as Rhom except the match is on the B fields.  Matched elements
        are said to be contextually related.  (SR # 3, page 49).
Rgen x
        All of the elements of the set of x are compared in pairs.
        If two elements of x have A field topics related by Simp (ie semantic
        implication, see SR # 3 page 46), the two elements are said to
        obey the generalizing relation and the (ordered) pair are put on the
        set of P with the more general member in the second position.
        [Note:  The semantic relations must be user created and in the
        appropriate format.]
Rexa x
        All elements of the set of x are compared in pairs.  If the one
        element of x has a B field topic and the other an A field
        topic related by Simp (semantic implication, see SR # 3 page 46),
        the one is said to be the explicational ampflication of the
        other and the (ordered) pair are put on the set of P with the
        element having the B field topic first.  [Again the semantic
        implication must be user created and in the appropriate format.]
Rmas x
        All the elements of the set of x are compared in pairs.  If
        two elements of x have A field topics related by Ssim (ie
        semantic similarity, see SR # 3 page 47), the two elements are
        said to be meaning associates and the (ordered) pair are put
        on the set of P.  [Note:  Semantic similarity must be user
```

```
        defined in the appropriate format.]

In each case the x stands for the name of some associated set.
To exit use 'halt  or hit the "rub out" key in response to any
question or request for input.
For more information see manual.
```

APPENDIX B

Examples of the User's Mode Dialogue with the STITE System

```
Name: ljq
Password:
Welcome to UNIX
%
```

This is the standard sign on through the UNIX operating system; to enter the User's Mode, one gives "user.a" ---

```
  user.a
        The Stite Users system is a yes-no driven, agenda type dialog
system.  Most requests for a user response are to be answered 'yes' or 'no
(or just 'y' or 'n').  When other type input is requested it should be clear
from the question what is wanted.  Input is always free field, never more than
one line at a time, and terminated by a carriage return.
        At various stages the system will attempt to inform you in some detail
. . do you wish to continue in this vein? (y or n)  n
Subject Matter and Topics
        The present system deals almost exclusively with the subject matter
Graph Theory and its applications.  Thus topics within this area will be
recognized.  Typically for example the topics
        simple graph              eulerian graph            digraph
        tree                      vertex                    mapping
. . do you wish to continue in this vein? (y or n)  n
  Do you wish to inquire about the subject matter in the system? (y or n) y
```

Here one has the option of inquiring about specific topics and subtopics in the data base, with the option of examining relevant items.

```
  In what  topic or subtopic are you interested?
duality
  The system has 3 entries related to {duality}
  do you wish to SEE some of them? (y or n) y
                A{duality}
        BIB{351}
TXT:
```

This is a bibliographic entry with "duality" in the A field.

```
duality and regge-mueller phenomenology for inclusive spectra
    freeman, j.r.,   univ. washington, seattle, usa
    nucl. phys. b (netherlands)(nupbbo) 18 march 1974,b71(2) 301-13 ,
     10 ref.
       physics abstracts(pa) 1974, 36096
    isc. a3568,a3430,a3450,   ish. elementary particle inclusive reac
       tions,elementary particle interaction models,duality and dual m
carriage return to continue . .
       odels,regge poles and trajectories,   iuc. gkemaw,ghgaac,ghmaaq
       ,
       keywds. duality,inclusive spectra,exchange degeneracy constrai
       nts,rapid scaling criterion,regge mueller phenomenology
abstr.. further predictions are examined of a recently proposed, phen
omenologically supported, duality scheme for double regge-mueller gra
phs. it is stressed that these duality constraints should be tested,
not only on the non-scaling behaviour at x#0 of single-particle spect
ra, but also on the shape (i.e. curvature) of rapidity spectra. excha
nge degeneracy constraints for single and double regge-mueller expans
carriage return to continue . .
ions are used, together with factorization to investigate two-particl
e spectra of (a#dcb) when one particle is near the kinematic bounda
ry and the other is in the central region. a rapid scaling criterion
is derived. the relative sizes of the non-scaling part of (p#pi/sup -
/cb) spectra c#pi/sup &/,pi/sup -/,pi/sup 0/, b#pi/sup &/,pi/sup -/
,k/sup &/,k/sup -/,p are estimated. an extended pattern of duality co
nstraints for many-particle regge-mueller graphs is suggested
  continue listing? (y or n) n
```

```
                B{duality}
        BIB{217}
```

This item has "duality" in the B field.

```
TXT:
a method of decomposition for large scale systems and its application
    tamura, h.,
    syst. and control (japan)(sycna9) june 1973,17(6) 352-60 , 31 ref
    . japanese
       computer & control abstracts(cca) 1974, 06610
    isc. c6410,c6630,c6280,   ish. large-scale systems,graph theory,d
      uality (mathematics),mathematical programming,optimal control,
carriage return to continue . .
 iuc. vccaag,vegaal,dlraap,
       keywds. method of decomposition,partitioning of large scale sy
       stems,structural partitioning,duality,mathematical programming
        systems,multistage decision processes,optimum control problem
       s,continuous systems,graph theory
abstr.. structural partitioning of large scale systems, graph-theory
structural partitioning, duality and partitioning of mathematical pro
gramming systems, multistage decision processes and partitioning syst
ems for optimum control problems in continuous systems are all dealt
with, with reference to block diagrams, structure-diagrams etc
  continue listing? (y or n) n
  Do you wish to enquire further about the subject matter? (y or n) y
```

Here one may inquire about other topics.

```
  In what other  topic or subtopic are you interested?
platonic graph
  The system has 7 entries related to {platonic graph}
  do you wish to SEE some of them? (y or n) y
                 A{platonic graph}
       MOD{575}
TXT: Of special interest among the regular graphs are the so-called
Platonic graphs, the graphs formed by the vertices and edges of the
five regular (Platonic) solids-the tetrahedron, cube, octahedron,
dodecahedron and icosahedron. The tetrahedral graph is K4 (see WIL/1/68),
and the graphs of the cube and octahedron are shown in WIL/1/72
and WIL/1/75.  We leave the drawing of the ocisahedral graph as an
excercise for the student.
                 B{platonic graph}
       MOD{696}
TXT: For which values of m and n are the following graphs Hamiltonian:
(i) Km,n;  (ii) Kn; (iii) Wn? Describe a Hamiltonian circuit in each
case.  Show also that the Platonic graphs are all Hamiltonian, and find
a Hamiltonian circuit for each.
       MOD{647}
  continue listing? (y or n) n
```

One has the option of discontinuing the item listings.

```
  Do you wish to enquire further about the subject matter? (y or n) y
  In what other  topic or subtopic are you interested?
vertex models
  The system has 1 entries related to {vertex models}
  do you wish to SEE some of them? (y or n) n
  Do you wish to enquire further about the subject matter? (y or n) y
  In what other  topic or subtopic are you interested?
eulerian graph
  The system has 29 entries related to {eulerian graph}
  do you wish to SEE some of them? (y or n) n
  Do you wish to enquire further about the subject matter? (y or n) n
```

Discontinue inquiring about topics.

```
        The major functions of the STITE system were broken down into
tasks and discussed in the STITE Report#2 (Dec 1973). Seventeen tasks were
identified and analyzed. All of these centered around the use of the
"internal" information, structured as modules and described in STITE Report #3.
        Data base elements are records of two kinds: modules of internal in-
formation (called moditems) and bibliographic records (called bibitems). The
. . do you wish to continue in this vein? (y or n)  n
At present the choice of tasks is as follows:
        Compilation of bibliographies on related topics (STITE Report #2,
        Dec 1973 pages 5 & 7, referred to there as Tasks No. 5 and 8).

        Compilation of module collections on related topics. (This includes
        the construction of outlines by topic or subtopic.)
. . do you wish to continue in this vein? (y or n)  n
  which task do you now wish.. 1&2
```

Here one arrives at the task options. The task options "1&2" gives a compilation of both internal and external modules related by a selected topic.

```
  what is the topic(s) of interest (if more than one separate by TABs)
graph
```

The topic "graph" was selected; there are 38 modules having graph as main topic (A field).

```
  do you wish a restriction?  (y or n) n
  there are 38, order 0 items;  do you wish to SEE some of them? (y or n) n
  do you wish to SAVE these items for future reference? (y or n) n
  do you wish to COUNT the order 1 items? (y or n) n


  continue this task with other topics? (y or n) y
```

Switch topics now to "directed graph".

```
  what is the topic(s) of interest (if more than one separate by TABs)
directed graph
  do you wish a restriction?  (y or n) n
  there are 7, order 0 items;  do you wish to SEE some of them? (y or n) n
```

There are 7 items related through A{directed graph}.

```
  do you wish to SAVE these items for future reference? (y or n) n
  do you wish to COUNT the order 1 items? (y or n) y
```

The order 1 items will be _explicationally_ related to the order 0 items.

```
  there are 13 topics,  do you wish to SEE some of them? (y or n) n
  there are 26, order 1 items;  do you wish to SEE some of them? (y or n) n
  do you wish to SAVE these items for future reference? (y or n) n
  do you wish to COUNT the order 2 items? (y or n) y
```

The order 2 items will be explicationally related to the order 1 items.

```
  there are 57 topics,  do you wish to SEE some of them? (y or n) n
  there are 83, order 2 items;  do you wish to SEE some of them? (y or n) n
  do you wish to SAVE these items for future reference? (y or n) n
  do you wish to COUNT the order 3 items? (y or n) n
```

Here we switch to another task; option 7 will allow selection of explicationally related bibliographic items.

```
  continue this task with other topics? (y or n) n
  which task do you now wish.. 7
```

Start with bibliographic item 474 (ie BIB{474}), which has three main topics (A field entries).

```
  what is the item number(s) of interest (if more than one separate by TABs)
474
  do you wish a restriction?  (y or n) n
  there are 3 topics,  do you wish to SEE some of them? (y or n) y

A{graph theory}
A{analysis of communication networks}
A{communication networks}
  there are 9, order 0 items;  do you wish to SEE some of them? (y or n) n
```

All of the order 0 items are related homonymicly, that is, have at least one common A field term. We choose to save these items here for future reference.

```
  do you wish to SAVE these items for future reference? (y or n) y
  by what name do you want these remembered
temp-1
  they have been added to temp-1
  do you wish to COUNT the order 1 items? (y or n) y
```

```
there are 41 topics,  do you wish to SEE some of them? (y or n) n
there are 24, order 1 items;  do you wish to SEE some of them? (y or n) n
do you wish to SAVE these items for future reference? (y or n) n
do you wish to COUNT the order 2 items? (y or n) n
Continue this task with other items? (y or n) n
which task do you now wish.. 13
```

Task option 1 finds paths or chains of terms that relate (explicationally) modules along the path. First a search is made to see if a path connects the given starting and ending terms. Then the option of exhibiting the terms along the path is given.

The restriction H{1} allows only those modules that are defintional to form links along the path.

```
give starting term:  directed graph
give ending term for path:  set
  do you wish a restriction?  (y or n) y
 What restriction . . (give as H{4-2} or D{2} or F{1}  etc)
H{1}
 no connection yet at level = 1 count = 7  continue search? (y or n) y
 no connection yet at level = 2 count = 7  continue search? (y or n) y
 connection at level = 3
  There are 14 terms at level 3;  do you wish to see them? (y or n) n
  Do you wish to construct paths? (y or n) y



                      3.1: B{set}
                 2.1: B{graph}
            1.1: B{digraph}
                      3.2: B{set}
                 2.2: B{graph}
            1.2: B{vertex}
0.2: A{directed graph}
       2  paths
  Do you wish to see ALL the items (modules or bibligraphic entries)
along ALL the paths? (y or n) n
```

```
  do not understand, say again . .
  which task do you now wish.. 13
give starting term:  eulerian graph
give ending term for path:  endpoint
  do you wish a restriction?  (y or n) y
 What restriction . . (give as H{4-2} or D{2} or F{1}  etc)
H{1}
```

Here is another example of an (explicational) path restricted to definitional (H{1}) modules.

```
 no connection yet at level = 1 count = 1  continue search? (y or n) y
 no connection yet at level = 2 count = 2  continue search? (y or n) y
 no connection yet at level = 3 count = 12  continue search? (y or n) y
 connection at level = 4
  There are 20 terms at level 4;  do you wish to see them? (y or n) n
  Do you wish to construct paths? (y or n) y



                                  4.1: B{endpoint}
                          3.1: B{linear graph}
                  2.1: B{circuit}
                                  4.2: B{endpoint}
                          3.2: B{graph}
                  2.2: B{edge}
          1.2: B{eulerian circuit}
0.2: A{eulerian graph}
      2  paths
  Do you wish to see ALL the items (modules or bibligraphic entries)
along ALL the paths? (y or n) n
```

Here one looks at the modules that form the connections along the path. Note here that MOD{434} connects "linear graph" and "end point", MOD{354} connects "circuit" and "linear graph", etc.

```
  Do you wish to examine individual paths? (y or n) y
  Give junction pair numbers (higher first, then lower). .
4.1,  1.2
                  INTER{A{linear graph}:B{endpoint}}
         MOD{434}
TXT: A linear graph corresponding to a set of edges is a linear graph
consisting of all edges and all vertices which are the endpoints of edges
in the set.
                              INTER{A{circuit}:B{linear graph}}
         MOD{354}
TXT: A circuit is a connected linear graph in which every vertex is of degree
2.
                  INTER{A{eulerian circuit}:B{circuit}}
         MOD{221}
TXT: An eulerian circuit of G is a circuit containing all the edges of G.

 do you want to see another individual path? (y or n) n
  Do you wish this path information remembered? (y or n) n
  which task do you now wish.. 13

give starting term:  arc
give ending term for path:        set
  do you wish a restriction?  (y or n) y
 What restriction . . (give as H{4-2} or D{2} or F{1}  etc)
H{1}
```

```
no connection yet at level = 1 count = 1  continue search? (y or n) y
no connection yet at level = 2 count = 7  continue search? (y or n) y
no connection yet at level = 3 count = 6  continue search? (y or n) y
connection at level = 4
 There are 14 terms at level 4;  do you wish to see them? (y or n) n
 Do you wish to construct paths? (y or n) y
```

Here are the paths connecting (explicationally) "arc" and "set".

```
                              4.1: B{set}
                      3.1: B{graph}
              2.1: B{digraph}
                              4.2: B{set}
                      3.2: B{graph}
              2.2: B{vertex}
      1.2: B{directed graph}
0.2: A{arc}
     2  paths
  Do you wish to see ALL the items (modules or bibligraphic entries)
along ALL the paths? (y or n) n
  Do you wish to examine individual paths? (y or n) y
  Give junction pair numbers (higher first, then lower). .
4.2,0.2
```

Here are the modules along the path:

```
              INTER{A{graph}:B{set}}
      MOD{532}
TXT: (FIG 532)
      MOD{47}
  continue listing? (y or n) y
TXT: Graph  A mathematical system consisting of two sets V and E,
together with a mapping ? of E into V & V (the set of unordered pairs
of elements of V).  The elements of V and E are called the vertices and
edges of the graph, respectively, and ? is called its incidence mapping.
If ?(e) = (v & w), vertices v and w are called the endpoints of edge e.
              INTER{A{vertex}:B{graph}}
      MOD{98}
TXT: Vertex  See Graph and Directed graph
              INTER{A{directed graph}:B{vertex}}
      MOD{186}
TXT: A directed graph or digraph is a finite nonempty set V (of vertices)
together with a set E (disjoint from V) of ordered pairs of distinct elements
of V.
      MOD{33}
TXT: Directed graph  A mathematical system consisting of two sets V and A,
together with a mapping ? of A into V  x V.  The elements  of V and A are
called the vertices and arcs of the directed graph, respectively, and ? is
called its directed incidence mapping.  If ?(a) = (v,w), v is called the
initial vertex of arc a and w is called its terminal vertex.
  continue listing? (y or n)Y
```

```
                INTER{A{arc}:B{directed graph}}
          MOD{6}
TXT: Arc  See Directed graph.

 do you want to see another individual path? (y or n) n
  Do you wish this path information remembered? (y or n) n
  which task do you now wish.. 13
```

Another explicational path:

```
give starting term:  simple chain
give ending term for path:  arc
  do you wish a restriction?  (y or n) y
 What restriction . . (give as H{4-2} or D{2} or F{1}  etc)
H{1}
 no connection yet at level = 1 count = 3  continue search? (y or n) y
 no connection yet at level = 2 count = 7  continue search? (y or n) y
 connection at level = 3
  There are 20 terms at level 3;  do you wish to see them? (y or n) n
  Do you wish to construct paths? (y or n) y
```

Here there is a single unique path from "simple chain" to "arc".

```
                        3.1: B{arc}
                  2.1: B{directed graph}
            1.1: B{vertex}
0.1: A{simple chain}
        1  paths
  Do you wish to see ALL the items (modules or bibligraphic entries)
along ALL the paths? (y or n) n
  Do you wish to examine individual paths? (y or n) n
  Do you wish this path information remembered? (y or n) n
  which task do you now wish.. 13
give starting term:  eulerian graph
give ending term for path:  arc
  do you wish a restriction?  (y or n) y
 What restriction . . (give as H{4-2} or D{2} or F{1}  etc)
H{1}
 no connection yet at level = 1 count = 1  continue search? (y or n) y
 no connection yet at level = 2 count = 2  continue search? (y or n) y
```

```
no connection yet at level = 3 count = 12  continue search? (y or n) y
no connection yet at level = 4 count = 20  continue search? (y or n) y
connection at level = 5
 There are 5 terms at level 5;  do you wish to see them? (y or n) n
 Do you wish to construct paths? (y or n) y
```

In this example the path structure is more complex; there are 4 paths connecting "eulerian graph" and "arc".

```
                                                5.1: B{arc}
                                     4.1: B{endpoint}
                          3.1: B{linear graph}
                                                5.2: B{arc}
                                     4.2: B{loop}
                          3.2: B{geometric graph}
                                                5.3: B{arc}
                                     4.3: B{directed graph}
                          3.3: B{vertex}
               2.3: B{circuit}
                                                5.4: B{arc}
                                     4.4: B{endpoint}
                          3.4: B{graph}
               2.4: B{edge}
          1.4: B{eulerian circuit}
0.4: A{eulerian graph}
       4  paths
  Do you wish to see ALL the items (modules or bibligraphic entries)
along ALL the paths? (y or n) n
  Do you wish to examine individual paths? (y or n) y
  Give junction pair numbers (higher first, then lower). .
5.2,2.3
                    INTER{A{loop}:B{arc}}
          MOD{60}
TXT: Loop  An edge (or arc) that is incident with only one vertex.
                    INTER{A{geometric graph}:B{loop}}
          MOD{45}
TXT: Geometric graph  A graph whose vertices are selected points in two-
or three-dimensional space and whose edges are nonintersecting simple curves
each of which joins two vertices (or, in the case of a loop, closes on a
single vertex) without containing any other vertices.
                    INTER{A{circuit}:B{geometric graph}}
          MOD{17}
TXT:  Circuit  A set of edges which, if properly ordered, form a circuit
progression.  In a geometric graph, a set of edges which form a closed curve.

 do you want to see another individual path? (y or n) n
  Do you wish this path information remembered? (y or n) n
  which task do you now wish.. 0
```

```
The % indicates you have returned to the UNIX operating system:
%
```

Task option 0 (Zero) indicates exit from the user mode of STITE and a return to the UNIX operating system.

APPENDIX C

The Data Base Manager's Instruction Dialogues for the STITE System

For carrying out certain tasks and for performing other desired operations, a command language for manipulating the records and creating structures is available.

Looking at the records or structures being manipulated, one notes that they are basiclly names and sets of names. Associated with each name is a set called its associated set. An associated set is just a set in that the order is not relevant and no duplicates are be allowed. Some of the associated sets are empty. Set theoretical operations are defined and implemented on these associated sets.

The commands are in a very simple form and restricted to unary and binary opperations. One of the operands is designated by a P or pointing register; the other, if there is one, is named in the command. (This is reminiscent of a single address machine language with an "accumulator" register, except that the "accumulator" register does not hold the item to be operated on but only points to it.) The item pointed to by the P or pointing register will sometimes be referred to as just P. Thus, the command, point X, puts P to X, that is, puts the P or pointer register pointing to X. The command attach Y adds Y to the P set, that is, adds the name Y to the associated set of the name pointed to by the P or pointer register.

The operations available are:

create x
Create the name "x", associate the null set, set P register pointing to x

point x
Set P register to x

attach x
Put x on the associated set of whatever is pointed by P register

remove x
Take x off the associated set of whatever is pointed by P

display name
Display name pointed by P

display set
Display the associated set of whatever is pointed by P

display secondary
Display the sets of the set of P

display n-ary N.m
Displays the sets of the sets of the . . (to a depth N) . . of P; if m = 1 only the first element on the Nth set will be displayed, if m = 0 all items on the Nth set will be displayed

add set x
Add the items on the x associated set to the items on the set pointed by P, eliminating duplicates (the union or OR operation)

delete set x
Delete from the P set all items on the pointed or P set (the AND NOT operation)

delete setnot x
Delete from the set pointed by P all items not on the x set (the intersection or AND operation)

count
Count the number of items on the pointed of P set, 0 if null

is member x
Responds yes if x is on the pointed (P) set, no otherwise

```
relation set x
        Gives set relation between the pointed set and the associated set
        of x.  Response:
                "identical" if x & P sets identical
                "superset" if x is a proper superset of P
                "subset" if x is a proper subset of P
                (count) of intersection of x & P
                "disjoint" if x & P sets disjoint
make set null
        Sets the P set to null
on secondary put x
        x is placed on all the sets of the items on the set of P
union secondary x
        The set of P is replaced by a set that is the union of the sets of
        the items on the x set
intersect secondary x
        The set of P is replaced by a set that is the intersection of
        the sets of items on x
exchange x
        Exchanges the sets of x and P;  x then has P's old set and
        P has the set of x
eliminate field F
        All the elements of the P set (that are of the type of the field
        F) are eliminated or removed;  F must be of the form A{ , B{, etc..
        or TXT:
eliminate fieldnot F
        All elements of the P set (that are not of the type of the field F)
        are removed from P;  F must be of the form A{, B{, etc.. or TXT:
ABiterate
        The set of P must consist of field name values (ie of the
        form B{ ... }).  The set of P is replaced as follows:
        1) convert all these B fields to A fields (with the same
        name values), 2) find all the modules for these A fields, 3)
        find all the B fields of all these modules.  The final P set is the
        set of all the B field name values obtained at step 3.
BAiterate
        Same as ABiterate except that the A and B fields are interchanged
BBiterate
        Same as ABiterate except that one starts with B fields (and ends with
        B fields)
ABconnect to x
        Performs ABiterate until a match on x is obtained;  after each
        iteration the number of items on the P set is given and the
        option of continuing or stopping is given
BAconnect to x
        Same as ABconnect except with A and B fields interchanged
BBconnect to x
        Same as ABconnect except that only B fields are used
ABpath to x
        The set of P must consist of only A field name values
        (ie of the form A{ ... }).  A search is made of the module sequences,
        using ABconnect, until a connection is made with x.  Before starting a
        request for the maximun depth of the search is made.  All paths for
        which a connection is found are printed with the depth of the
        connection specified.
BApath to x
```

```
        Same as ABpath with A and B fields interchanged
BBpath to x
        Same as ABpath except that only B fields are used
Rexp x
        All of the elements of the set of x are compared in pairs.  If two
        elements of x have common topic (name value) on their sets that is in
        the A field of one and in the B field of the other,  the two elements
        are said to be explicationally related and the (ordered) pair are put
        on the set of P with the member having the A field match first.
        (SR # 3, page 48).
Rhom x
        All elements of the set of x are compared in pairs.  If two elements
        of x have the common topic in an A field of their sets, the two
        elements of x are said to be homonomicly related and the (ordered) pair
        are put on the set of P.  (SR # 3, page 51).
Rcon x
        Same as Rhom except the match is on the B fields.  Matched elements
        are said to be contextually related.  (SR # 3, page 49).
Rgen x
        All of the elements of the set of x are compared in pairs.
        If two elements of x have A field topics related by Simp (ie semantic
        implication, see SR # 3 page 46), the two elements are said to
        obey the generalizing relation and the (ordered) pair are put on the
        set of P with the more general member in the second position.
        [Note:  The semantic relations must be user created and in the
        appropriate format.]
Rexa x
        All elements of the set of x are compared in pairs.  If the one
        element of x has a B field topic and the other an A field
        topic related by Simp (semantic implication, see SR # 3 page 46),
        the one is said to be the explicational ampflication of the
        other and the (ordered) pair are put on the set of P with the
        element having the B field topic first.  [Again the semantic
        implication must be user created and in the appropriate format.]
Rmas x
        All the elements of the set of x are compared in pairs.  If
        two elements of x have A field topics related by Ssim (ie
        semantic similarity, see SR # 3 page 47), the two elements are
        said to be meaning associates and the (ordered) pair are put
        on the set of P.  [Note:  Semantic similarity must be user
        defined in the appropriate format.]

In each case the x stands for the name of some associated set.
To exit use 'halt  or hit the "rub out" key in response to any
question or request for input.
For more information see manual.
```

APPENDIX D

Examples of the Data Base Managers Mode and Command Language for the STITE System

The Data Base Manager's Mode is entered by calling the program "inter.a".

```
% inter.a

This is the interactive manipulator for association lists.
        For carrying out certain tasks and for performing other desired
operations, a command language for manipulating the records and creating
structures is available.
        Looking at the records or structures being manipulated, one notes that
they are basiclly names and sets of names.  Associated with each name is a
. . do you wish to continue in this vein? (y or n)  y
set called its associated set.  An associated set is just a set in that
the order is not relevant and no duplicates are be allowed.  Some
of the associated sets are empty.  Set theoretical operations are defined
and implemented on these associated sets.
        The commands are in a very simple form and restricted to unary and
binary opperations.  One of the operands is designated by a P or pointing
register;  the other, if there is one, is named in the command.  (This is
reminiscent of a single address machine language with an "accumulator"
register, except that the "accumulator" register does not hold the item
to be operated on but only points to it.)  The item pointed to by the P
or pointing register will sometimes be referred to as just P.  Thus, the
command, point X, puts P to X, that is, puts the P or pointer register
pointing to X. The command attach Y adds Y to the P set, that is, adds
the name Y to the associated set of the name pointed to by the P or
pointer register.
. . do you wish to continue in this vein? (y or n)  n
Make timing measurements? (y or n) n
enter requests:
>point MOD{26}
>display set
```

The full set of instructions available in this mode are described in Appendix... The symbol ">" requests input.

```
MOD{26}
TXT: Cut-set  A disconnecting set consisting of all the edges that join
a specified set of vertices with the complementary set of vertices.
B{complementary set of vertices}
C{BUS /1/26}
H{1}
F{1}
```

Note that when a module is displayed all of the fields (attributes) are exhibited also.

```
A{cut-set}
B{edge}
B{connectivity}
B{vertex}
D{1}
>point A{cut-set}
>count
  5
>display set
A{cut-set}
MOD{170}
MOD{640}
MOD{644}
  continue listing? (y or n) y
MOD{370}
MOD{26}
>point B{edge}
>count
  137
>point B{conectivity}
```

("connectivity" misspelled.)

```
that name does not yet exist in file;  can not point to it
>point B{connectivity}
>count
  4
>display secondary
```

```
                B{connectivity}
        MOD{24}
TXT: Component of a graph  A connected subgraph which is not contained in any
larger connected subgraph.
A{graph component}
B{subgraph}
H{1}
C{BUS /1/24}
F{1}
B{connectivity}
D{1}
        BIB{657}
TXT:
optimization of symmetric sparse matrices
    enekes, j.,venetsanopoulos, a.n.,black, b.a.,   univ. toronto, on
    tario, canada
    optimization of symmetric sparse matrices, 7th asilomar conferenc
    e on circuits, systems and computers 27-29 nov. 1973, pacific gro
carriage return to continue . .y
    ve, calif., usa, western periodicals, north hollywood, calif., us
    a, 1974 677-81 , 8 ref.
       physics abstracts(pa) 1974, 59037 and electrical & electronics
        abstracts(eea) 1974, 29415 and computer & control abstracts(c
       ca) 1974, 18553
    isc. a1110,b1410,c6210,   ish. matrix algebra,graph theory,   iuc
      . dbccms,dlcaaq,
       keywds. symmetric sparse matrices,l-u factorization,equivalent
        graph problem,connectivity,block minimal separator tree
abstr.. the problem of l-u factorization of a symmetric sparse matrix
 is transformed into an equivalent graph problem. it is shown that in
 order to minimize the 'fill-in' during the l-u factorization, it is
sufficient to examine the connectivity of the graph corresponding to
the sparse matrix. a method of optimally labelling the graph in terms
 of the so-called block-minimal separator tree is proposed
  continue listing? (y or n) y
B{equivalent graph problem}
B{block minimal separator tree}
B{matrix algebra}
A{symmetric sparse matrices}
B{graph theory}
B{l-u factorization}
B{connectivity}
        BIB{125}
TXT:
dividing a graph into triconnected components
carriage return to continue . .
    hopcroft, j.e.,tarjan, r.e.,   cornell univ., ithaca, n.y., usa
    siam j. comput. (usa)(smjcat) sept. 1973,2(3) 135-58 , 19 ref.
       electrical & electronics abstracts(eea) 1974, 05332 and comput
       er & control abstracts(cca) 1974, 03727
    isc. b1480,c6270,   ish. graph theory,   iuc. dlpaaz,
       keywds. dividing,graph,triconnected components,algorithm,artic
       ulation point,connectivity,separability
abstr.. an algorithm for dividing a graph into triconnected component
s is presented. when implemented on a random access computer, the alg
orithm requires o(v&e) time and space to analyze a graph with v verti
carriage return to continue . .
ces and e edges. the algorithm is both theoretically optimal to withi
n a constant factor and efficient in practice
  continue listing? (y or n) y
B{articulation point}
B{graph theory}
A{dividing}
```

There are four modules having "connectivity" in their B field; two of these are internal and two are external (bibliographic) modules.

```
A{graph}
A{triconnected components}
B{separability}
B{algorithm}
B{connectivity}
        MOD{26}
TXT: Cut-set  A disconnecting set consisting of all the edges that join
carriage return to continue . .
a specified set of vertices with the complementary set of vertices.
B{complementary set of vertices}
C{BUS /1/26}
H{1}
F{1}
A{cut-set}
  continue listing? (y or n) y
B{edge}
B{connectivity}
B{vertex}
D{1}
```

Create a new name "templ0" for convenience; the remaining instructions give examples of the kinds of operations available; for a description of these commands see Appendix C

```
>create templ0
>add set B{edge}
>count
  137
>is member MOD{26}
  yes
>delete setnot B{connectivity}
>display set
templ0
MOD{26}
>relation set A{cut-set}
  superset
>eliminate field MOD{
>count
  0
>add set A{cut-set}
>count
  5
>display          list
  do not understand, say again . .
>display set
templ0
MOD{170}
MOD{640}
MOD{644}
MOD{370}
MOD{26}
>intersect secondary templ0
>count
  3
>display set
templ0
H{1}
F{1}
A{cut-set}
  continue listing? (y or n) y
>  halt
$
```

The "halt" takes one back to the Unix operating systems.

APPENDIX E

The Program Listings for the STITE System

```
#define getcr while (getchar() != 012)



/*      u.c

/*      u.c contains some utility routines needed by
/*      most programs.

*/

char cont[]{"..continue? "};
char contin[]{
". . do you wish to continue in this vein? (y or n)   "};

error(c)  char *c;
{while (1)
  {printf("fatal ERROR \n");
  printf(c);
  getcr;}
}/*end of error(c) */

int geta(y)  int y;
{int c;
while ((c = getchar()) <= ' ');
getcr;
return(y == c);
}/*end ot geta(y)*/

/*
getchar()
{int c;
if (read(0, &c, 1) <= 0) return(0);
return(c);
}/*end of getchar()*/

int readint(c)  int c;
{int j;
while (c > 071 || c < 060) c = getchar();
j = 0;  while (c <= 071 && c >= 060)
  {j = j * 10 + (c - 060);
  c = getchar();}
return(j);
}/*end of readint(c)*/

int eql(s1, s2)  char *s1, *s2;
{int c;
while((c = *s1++) == (*s2++)) if (c == 0) return(1);
return(0);
}/* end of  eql(s1, s2)*/

move(s0, s, j)  char *s0, *s;  int j;
{char *sj;
sj = s0 + j;
while (*s++ = *s0++)
```

```
if (s0 >= sj) {*(--s) = 0; return;}
}/*end of move(s0, s, j)*/

int equal(s1, s2, j)  char *s1, *s2;  int j;
{while (j--)
if (*s1++ != *s2++) return(0);
return(1);
}/*end of int equal(s1, s2, j)*/

int describe(file)  char *file;
{static char *f;  static int fl, q;
int j, k, c;
c = 0;  j = 15;
if (f != file)
        {f = file;  j = 6;  q = 0;
        fl = open(file, 0);
        seek(fl, 0, 0);}
/*for debug do*/
if (fl <= 0)
        {printf(file);
        printf("\n fl = %d  in describe of u.c ", fl);}
/*end debug*/
k = 1;  q =+ 1;
while (j && k)
        {k = read(fl, &c, 1);
        if (c == 012) j =- 1;
        putchar(c);}
if (!k) {close(fl);  f = 0;  return(0);}
if (q < 3) printf(contin); else printf(cont);
if (geta('y')) return(1);
close(fl);  f = 0;  return(0);
}/*end of describe(file)*/

int getstr(s, k, ch1)  char s[];  int k, *ch1;
{int j;
s[0] = 0;  while (s[0] <= ' ') s[0] = getchar();
j = 0;  while (s[j] >=    )
        {if (j < k-1) j =+ 1;  s[j] = getchar();}
*ch1 = s[j];
while (s[--j] <= ' ');
s[++j] = 0;
return(j);
}/*end of getstr(s, k, ch1)*/

int getword(s, k, ch1)  char s[];  int k, *ch1;
{int j;
s[0] = 0;  while (s[0] <=    ) s[0] = getchar();
j = 0;  while (s[j] >    )
        {if (j < k-1) j =+ 1;  s[j] = getchar();}
*ch1 = s[j];
s[j] = 0;
return(j);
}/*end of getword(s, k, ch1)*/

concat(ch1, ch2, j1)  char *ch1, *ch2;  int j1;
{static char *ch;  char *chover;
```

Feb 5 12:30 u.c Page 3

```
if (!*chl) ch = chl;
chover = chl + jl;
while ((ch < chover)&&(*ch++ = *ch2++));
ch =- 1;  *ch = 0;
}/*end of concat(chl, ch2, jl)*/

printab(n)  int n;
{while (0 < n--) putchar(011);
}/*end of printab(n)*/

char *alfize(a, n)  int n;  char *a;
{int k, j, temp;
if (!n) {a[0] = '0';  a[1] = 0;  return(a);}
k = 10000;
j = 0;  while (k)
        {temp = n/k;  n =% k;  k =/ 10;
        if (temp || j) a[j++] = temp + 060;}
a[j] = 0;
return(a);
}/*end of alfize(a, n)*/
```

```
#define nbuf 48
#define h2mod 73
#define getcr while (getchar() != 012)
/*h2mod should be a prime number about 50% greater than nbuf*/


/*      lp.c

/*      lp.c contains a collection of procedures for the
/*      linked list system (links in "links.d")

*/



extern error();
extern int errno;
int pgfnu, pgrcl, debug01, fllinks;
int lnfnu, lnrcl, flg01, *lst01;
struct quads{int x0[4]; char y0[4];};
struct bufferl
{int writeflg, pgfnu0, pgrcl0;
char lnfnu0, lnrcl0: struct quads q0[42];};
struct quads *qp; int *xp;  char  *yp;
struct bufferl buf0[nbuf], *bufp, *bufn;
struct ab {struct ab *a, *b; int flpg;};
struct ab ab0[nbuf+1];
struct ab *abp, *abn;
struct h2
{int filepage; struct h2 *suc;
struct bufferl *bufpage;};
struct h2 *h2p, *h2n, h20[nbuf], *h2recl, *start0[h2mod];

gtpgfnu()
{return(pgfnu);
}/*end of gtpgfnu()*/

read512(fl, buffer, n) int fl, n; char buffer[];
{int cnt;
cnt = read(fl, buffer, n);
if (n==512 && cnt==n && !errno) return;
printf("\n errno = %d cnt = %d n = %d pgfnu = %d lnfnu = %d ",
errno, cnt, n, pgfnu, lnfnu);
error("read error in link file;  see read512 (of lp.c)");
}/*end of read512(fl, buffer, n);*/

lnkinit()
{int j, jpage, jline;
fllinks = open("/mnt/stite/tf5.links.d", 2);
jpage = nbuf; while (jpage--)
  {bufp = buf0 + jpage;
  seek(fllinks, jpage, 3); /*3 means offset is times 512 */
  read512(fllinks, bufp, 512);
  bufp -> writeflg = 0;}
```

```
bufp = buf0;
pgfnu = buf0 -> pgfnu0;
pgrcl = buf0 -> pgrcl0;
lnfnu = buf0 -> lnfnu0;
lnrcl = buf0 -> lnrcl0;
abn = ab0 + nbuf;
abp = ab0 - 1;  while (++abp <= abn)
  {abp -> a = abp - 1;  abp -> flpg = abp - ab0;  abp -> b = abp + 1;}
abn -> b = ab0;
ab0 -> a = abn;
abn -> flpg = (-1);
j = nbuf;  while (j--)
  {h2p = h20 + j;
  start0[j] = h2p;
  h2p -> filepage = j;
  h2p -> suc = h20 - 1;
  h2p -> bufpage = buf0 + j;}
j = nbuf - 1;  while (++j < h2mod)
  {start0[j] = h20 - 1;}
h2recl = h20 - 1;
bufn = buf0 + nbuf;
h2n = h20 + nbuf;
} /*end of lnkinit()*/

/*verificacion of hash(02) tables & pointers */
verify()
{int j;
j = (-1); while (++j < nbuf)
  {h2p = h20 + j;
  abp = ab0 + j;
  printf("%d    %d %d %d      %d %d %d \n",
  start0[j] - h20,
  h2p -> filepage,
  h2p -> suc - h20,
  h2p -> bufpage - buf0,
  abp -> a - ab0,
  abp -> b - ab0,
  abp -> flpg);}
j = nbuf - 1;  while (++j < h2mod)
  {printf("%d \n", start0[j] - h20);}
printf("%d %d %d %d %d %d %d %d %d \n",
h20, h2n,
buf0, bufn,
start0,
h2recl - h20,
abn -> a - ab0,
abn -> b - ab0,
abn -> flpg);
printf("%d %d %d %d \n",
pgfnu, lnfnu, pgrcl, lnrcl);
printf("end of verify");
getcr;
}/*end of verify()*/

verpage()
/*verification of page(bufp)*/
```

```
{int l, jline;
jline = (-1);  while (++jline < 42)
  {qp = bufp -> q0 + jline;
  printf( "\n");
  l = (-1);  while (++l < 4)
    {printf("%d %d ",
    qp -> x0[l],
    qp -> y0[l]);}}
printf("\n end of verpage");  getcr;
}/*end verpage()*/

shift() /*abp is marked 'most recent referenced'*/
{struct ab *bn, *ap, *bp, *l;
if (abn -> b != abp)
  {bn = abn -> b;
  ap = abp -> a;
  bp = abp -> b;
  abp -> b = bn;
  abn -> b = abp;
  ap -> b = bp;
  abp -> a = abn;
  bn -> a = abp;
  bp -> a = ap;}
/*for debug do*/
if (debug01)
  {l = ab0 - 1;  while (++l <= abn)
    {printf("\n %d %d %d",
    l -> a - ab0,
    l -> b - ab0,
    l -> flpg);}
  printf("\n end of shift");  getcr;}
/*end of debug*/
}/*end of shift()*/

getlf(page)  int page;
/*finds position of page in h2*/
{lst01 = &(start0[page%h2mod]);
h2p = *lst01;
flg01 = 1;  while (h2p >= h20 && flg01)
  {if (page == h2p -> filepage)
  flg01 = 0;
  else
    {lst01 = &(h2p -> suc);
    h2p = *lst01;}}
/*for debug do*/
if (debug01)
  {printf("\n end of getlf %d %d",
  h2p - h20,
  flg01);  getcr;}
/*end of debug*/
/*flg01 == 0 if page is found,
h2p -> filepage == page if page is found, otherwise h2p == h20-1 */
}/*end of getlf(page, lst01, flg01)*/

removeh2(page)  int page;
/*removes page from hash table 2*/
```

```
{getlf(page);
/*flg01 == 0 if page is found, h2p == h20-1 if page not found */
if (flg01)
error("asked to remove page not present, see removeh2(page)");
else
  {h2recl = h2p;
  *lst01 = h2p -> suc;}
/*for debug do*/
if (debug01)
  {printf("\n end of removeh2 %d %d %d",
  h2p - h20,
  h2p -> suc - h20,
  flg01);  getcr;}
/*end debug*/
}/*end of removeh2(page)*/

enterh2(page)  int page;
/*puts page in hash table 2*/
{getlf(page);
/* flg01 == 0 if page is found, h2p == h20-1 if page is not found*/
if (!flg01)
error("asked to enter page already there, see enterh2(page)");
else
  {*lst01 = h2p = h2recl;
  if (h2p >= h2n || h2p < h20)
  error("h2p out of range in enterh2(page)");
  h2recl = h20 - 1;
  h2p -> suc = h20 - 1;
  h2p -> filepage = page;
  h2p -> bufpage = bufp;}
/*for debug do*/
if (debug01)
  {printf("\n end of enterh2 %d %d %d",
  h2p - h20,
  bufp - buf0,
  flg01);  getcr;}
/* end debug*/
}/*end of enterh2(page)*/

rollin(page)  int page;
/*rolls out old page and rolls page in*/
{abp = abn -> a;
bufp = buf0 + (abp - ab0);
if (bufp -> writeflg)                          /*note no semicolon*/
  {seek(fllinks, abp -> flpg, 3);
  write(fllinks, bufp, 512);}
removeh2(abp -> flpg);
if (lnfnu != 0 || page < pgfnu)          /*note no semicolon*/
  {seek(fllinks, page, 3);
  read512(fllinks, bufp, 512);}
/*for debug do*/
if (debug01)
  {printf("\n this is rollin %d %d %d",
  bufp - buf0,
  abp - ab0,
  abp -> flpg);  getcr;}
```

```
/*end debug*/
abp -> flpg = page;
bufp -> writeflg = 0;
shift();
enterh2(page);  /*bufp is set*/
}/*end of rollin(page)*/

setpage(page)  int page;
/*sets bufp to page requested*/
{static int j;
h2p = start0[page%h2mod];
bufp = buf0 - 1;  while(h2p >= h20 && bufp < buf0)
  {if (page == h2p -> filepage)
  bufp = h2p -> bufpage;
  else
  h2p = h2p -> suc;}
/*for debug do*/
if (debug01)
  {printf("\n this is setpage %d %d %d ",
  h2p - h20,
  bufp - buf0,
  h2p -> suc - h20);  getcr;}
/*end debug*/
if (bufp >= buf0)
  {abp = ab0 + (bufp - buf0);
  if (!(j = (j+1)%3))
  shift();}
else rollin(page);
}/*end of setvpage(page)*/

setlink(line, p0, 10, pl, 11, p2, 12, p3, 13)
int line, p0, 10, pl, 11, p2, 12, p3, 13;
{qp = bufp -> q0 + line;
xp = qp -> x0;
yp = qp -> y0;
if (bufp < buf0 ¦¦ bufp >= bufn)
error("bufp out of range in setlinks(line, p0, ..  etc)");
if (line < 0 ¦¦ line >= 42)
        {printf("line = %d ", line);
        error("in setlink(line, p0 .. etc) of lp.c");}
if (p0 > -2) {*(xp  ) = p0;  *(yp  ) = 10;}
if (pl > -2) {*(xp+1) = pl;  *(yp+1) = 11;}
if (p2 > -2) {*(xp+2) = p2;  *(yp+2) = 12;}
if (p3 > -2) {*(xp+3) = p3;  *(yp+3) = 13;}
bufp -> writeflg = 1;
}/*end of setlink(line, p0,10, pl, 11, p2, 12, p3, 13)*/

getlink(line, x, y)
int line, x[],y[];
{int l;
qp = bufp -> q0 + line;
xp = qp -> x0;
yp = qp -> y0;
l = 4;  while (l--)
  {x[l] = xp[l];  y[l] = yp[l];}
}/*end of getlink(line, ...)*/
```

```
/*this version of reclaim does NOT sort the free links*/
reclaim(page, line)  int page, line;
{if (page < 0)
        {printf("\n page = %d ", page);
        error("page < 0 in reclaim (of lp.c)");}
setpage(page);
setlink(line,
-1, -1, -1, -1, -1, -1,
pgrcl, lnrcl);
pgrcl = page;
lnrcl = line;
}/*end of reclaim(page, line)*/

newlink(pl)  int pl[];
/*pl[0] recieves the page, pl[1], the line*/
{int x[4], y[4];
if (pgrcl > -1)
  {pl[0] = pgrcl;
  pl[1] = lnrcl;
  setpage(pl[0]);
  getlink(pl[1], x, y);
  pgrcl = x[3];
  lnrcl = y[3];}
else
  {pl[0] = pgfnu;
  pl[1] = lnfnu;
  setpage(pgfnu);
  if (++lnfnu >= 42)
    {lnfnu = 0;
    pgfnu =+ 1;}}
/*for debug do*/
if (debug01)
  {printf("\n end newlink %d %d %d %d",
  pgfnu,
  lnfnu,
  pgrcl,
  lnrcl);  getcr;}
/*end debug*/
}/*end of newlink(pl)*/

flushln()
{int j;
setpage(0);
bufp -> writeflg = 1;
bufp -> pgfnu0 = pgfnu;
bufp -> lnfnu0 = lnfnu;
bufp -> pgrcl0 = pgrcl;
bufp -> lnrcl0 = lnrcl;
j = nbuf;  while(j--)
  {abp = ab0 + j;  bufp = buf0 + j;
  if (bufp -> writeflg)                          /*note no semicolon*/
    {seek(fllinks, abp -> flpg, 3);
    /*for debug do*/
    if (debug01) printf("%d ", abp -> flpg);
    /*end debug*/
```

```
    bufp -> writeflg = 0;
    write(filinks, bufp, 512);}}
}/*end of flushbuf()*/
```

```
/*      cf.c

/*      cf.c contains proceedures for manipulating
/*      the character files (flch: "/dev/tf5")

*/


extern int errno;
int jch, pagech;  char chbuf[512], chbufx[512];
int debug02, flch;
extern error();

chinit()
/*initializes the character file*/
{int cnt[2];
flch = open("/dev/tf5", 2);
seek(flch, 0, 0);
if (4 != read(flch, cnt, 4) ||
errno) error(
"read error in chinit of cf.c");
pagech = cnt[0];
jch = cnt[1];
}/*end of chinit*/

getjch()
{return(jch);
}/*end of getch()*/

getpgch()
{return(pagech);
}/*end of getpgch()*/

writech(c)  int c;
/*write a single character*/
{static int j;
chbuf[j] = c;
if (c == 0 && jch%2 == 0)
  {jch =+ 1;
  chbuf[++j] = c;}
if (c == 0 || jch+1 >= 512)
  {seek(flch, pagech, 3);
  seek(flch, jch-j, 1);
  write(flch, chbuf, j+1);
  j = 0;}
else j =+ 1;
/*for debug do*/
if (debug02)
printf("\n %c %d %d %d ",
c, pagech, jch, j-1);
/*end debug*/
```

```
if(++jch >= 512) {pagech =+ 1; jch = 0;}
return(c);
}/*end of writech(c)*/

charead(fl, cbuf, n) int fl, n;  char cbuf[];
{int cnt;
cnt = read(fl, cbuf, n);
if(cnt && !errno && n == 512) return;
printf("\n errno = %d cnt = %d n = %d  %d.%d",errno,cnt,n,pagech,jch);
error("read error in character file;  see charead (of cf.c)");
}/*end of charead(fl, cbuf, n)*/

rdchfl(pc, cbuf, fl)  int fl, pc[]; char cbuf[];
/*read a single character */
/*pc[0] is the page, pc[1] the character count*/
{if (pc[1] >= 512)
  {pc[0] =+ 1;
  pc[1] =- 512;
  seek(fl, pc[0], 3);
  charead(fl, cbuf, 512);}
/*for debug do*/
if (debug02)
printf("\n %c %d %d ",
cbuf[pc[1]], pc[0], pc[1]);
/*end debug*/
return(cbuf[pc[1]++]);
}/*end of rdchfl(pc,cbuf)*/

int strngpr(page, j, fl, fc, lc)  int page, j, fl, fc, lc;
{int pc[2], c;  static int jcr;
if (jcr > 15) jcr = 0;
putchar(fc);
if (fc == 012) jcr =+1;
pc[0] = page - 1;
pc[1] = j + 512;
while(c = rdchfl(pc, chbufx, fl))
        {putchar(c);
        if (c == 012)
        if (++jcr == 10 || jcr == 20 || jcr == 40)
                {printf("carriage return to continue . .");
                while (getchar() != 012);}}
putchar(lc);
if (lc == 012) jcr =+ 1;
return(jcr);
}/*end of strngpr(page, j, fl, fc, lc)*/

int prstrng(page, j)  int page, j;
{return(strngpr(page, j, flch, 0, 012));
}/*end of prstrng(page, j)*/

int prntstr(page, j, fc, lc)  int page, j, fc, lc;
{return(strngpr(page, j, flch, fc, lc));
}/*end of prntstr(page, j, fc, lc)*/

strngeq(page1, j1, fl1, page2, j2, fl2)  int page1, j1, fl1, page2, j2, fl2;
{int pc1[2], pc2[2], c1, c2;
```

```
pc1[0] = page1 - 1;
pc2[0] = page2 - 1;
pc1[1] = j1 + 512;
pc2[1] = j2 + 512;
do{c1 = rdchfl(pc1, chbuf, fl1);
  c2 = rdchfl(pc2, chbufx, fl2);
  /*for debug do*/
  if (debug02) printf("\n %c %c ", c1, c2);
  /*end debug*/}
while (c1==c2 && c1);
return (c1==c2);
}/*end of strngeq*/

readch(pc) int pc[];
/*read a single character from flch*/
{if (pc[1] >= 512)
  {pc[0] =+ 1;
  pc[1] =- 512;
  seek(flch, pc[0], 3);
  charead(flch, chbufx, 512);}
return(chbufx[pc[1]++]);
}/*end readch(pc)*/

chareq(page, j, chstr)  int page, j;  char chstr[];
{int k, pc[2], c;
pc[0] = page - 1;
pc[1] = j + 512;
k = (-1);
do {c = readch(pc);  k =+ 1;}  while(c == chstr[k] && c);
return(c == chstr[k]);
}/*end of chareq(page, j, chstr) */


strget(page, j, str, n)  int page, j, n;  char str[];
{int k, pc[2], c;
pc[0] = page - 1;
pc[1] = j + 512;
k = 0;  c = 1;
while (c && k < n)
str[k++] = c = readch(pc);
str[n-1] = 0;
}/*strget(page, j, str, n)*/

flushch()
{static int cnt[2];
if (cnt[1] == jch && cnt[0] == pagech) return;
cnt[0] = pagech;
cnt[1] = jch;
seek(flch, 0, 0);
write(flch, cnt, 4);
/*for debug do*/
if (debug02) printf("\n in flushch %d %d  ", pagech,jch);
/*end debug*/
}/*end of flushch()*/
```

```
#define lmax 128
#define h1m 3
#define h1mline 41
#define getcr while (getchar() != 012)
#define newk (if(tot)tot=/3;else tot=(++j*ll)^ip;k=1+tot%3;)

/*h1m should be prime and different from h1mline;
/*h1mline should be less than or equal to 42 and preferably prime;
/*and both should agree with create.c
*/



/*      h1.c

/*      h1.c contains procedures for the hash01 algorithm

*/



extern flushin();       extern error();         extern strget();
extern setlink();
extern setpage();
extern getlink();
extern newlink();
extern flushch();       extern getpgch();       extern getjch();
extern writech();       extern chareq();        extern prstrng();
int debug02;

getname(line, x, y)  int line, x[], y[];
(getlink(line, x, y);
y[0] = (y[0] << 1) & 0777;
)/*end of getname(line, x, y)*/

putat(page, line, chstr) int page, line;  char chstr[];
(int j;
/*setpage(page);*/  /*page should be set*/
setlink(line, getpgch(), getjch()>>1, -1, -1, -1, -1, -1, -1);
j = 0;  while (writech(chstr[j++]));
)/*end of putat(page, line, chstr))*/

int hash01(chstr, pl)  int pl[];  char chstr[];
(int c1, c2, page, line, j, k, tot, x[4], y[4];
tot = chstr[0];
j = k = 0;  while((c1=chstr[j++]) && (c2=chstr[j++]))
tot =^ (c1<<7) ^ c2;
x[k] = tot%h1m;
y[k] = tot%h1mline;
while (1)
  (page = pl[0] = x[k];
  line = pl[1] = y[k];
  /*for degug do*/
  if (debug02)
    (printf("\n this is hash01 %d %d ",
```

```
        page, line);  getcr;}
      /*end of debug*/
      setpage(page);  getname(line, x, y);
      if (x[0] < 0)
        {putat(page, line, chstr);  return(0);}
      /*returns 0 if chstr was absent*/
      if (chareq(x[0], y[0], chstr)) return(1);
      /*returns 1 if chstr was already present */
      tot =>> 1;  k = 2 + tot%2;
      if (x[k] < 0)
        {newlink(pl);
        putat(pl[0], pl[1], chstr);
        x[k] = pl[0];  y[k] = pl[1];
        setpage(page);
        setlink(line, -2, -2, -2, -2, x[2], y[2], x[3], y[3]);
        setpage(pl[0]);  return(0);}}
    /*returns 0 if chstr was absent, otherwise 1 */
    }/*end of hash01(chstr, pl)*/

    int hashno(chstr, pl)  int pl[];  char chstr[];
    {int c1, c2, page, line, j, k, tot, x[4], y[4];
    tot = chstr[0];
    j = k = 0;  while((c1=chstr[j++]) && (c2=chstr[j++]))
    tot =^ (c1<<7) ^ c2;
    x[k] = tot%h1m;
    y[k] = tot%h1mline;
    while (1)
      {page = pl[0] = x[k];
      line = pl[1] = y[k];
      /*for degug do*/
      if (debug02)
        {printf("\n this is hashno %d %d ",
        page, line);  getcr;}
      /*end of debug*/
      setpage(page);  getname(line, x, y);
      if (x[0] < 0) return(0);
      /*returns 0 if chstr was absent*/
      if (chareq(x[0], y[0], chstr)) return(1);
      /*returns 1 if chstr was already present */
      tot =>> 1;  k = 2 + tot%2;
      if (x[k] < 0) return(0);}
    /*returns 0 if chstr was absent, otherwise 1 */
    /*note: if chstr is absent hashno does NOT put it there! */
    }/*end of hashno(chstr, pl)*/

    fetchst(pl, str, n)  int n, pl[];  char str[];
    {int x[4], y[4];
    setpage(pl[0]);  getname(pl[1], x, y);
    strget(x[0], y[0], str, n);
    }/*end of fetchst(pl, str, n)*/

    cnvrtfld(ystr, str)  char ystr[], str[];
    {char strtemp[lmax];  int j, k, c;
    j = (-1);  while ((strtemp[++j] = c = str[j])  && j < lmax - 1)
    if (c == '(') k = j + 1;
    j = (-1);  while ((str[++j] = ystr[j]) && j < lmax - 1);
```

```
while ((str[j++] = strtemp[k++]) && j < lmax);
str[lmax-1] = 0;
}/*end cnvrtfld(ystr, str)*/

nnt xgetsy(plx, y, ply)  int plx[], ply[];  char y[];
{char str[lmax];
fetchst(plx, str, lmax);
cnvrtfld(y, str, ply);
return(hashno(str, ply));
}/*end of xgetsy(plx, y, ply)*/

opname(namex, namey, nameop, op)
int namex[], namey[], nameop[];  char op[];
{char strx[lmax], stry[lmax], str[lmax], *strj;
int j, x[4], y[4];
setpage(namex[0]);  getname(namex[1], x, y);
strget(x[0], y[0], strx, lmax);
setpage(namey[0]);  getname(namey[1], x, y);
strget(x[0], y[0], stry, lmax);
j = (-1);  while (str[++j] = op[j]);  j =- 1;
strj = strx;  while ((str[++j] = *strj++) && j < lmax-4);
str[j] = ':';
strj = stry;  while ((str[++j] = *strj++) && j < lmax-3);
str[j] = ')';
str[j+1] = 0;
/*for debug do
printf("\n");  printf(str);
/*end of debug*/
hash01(str, nameop);
}/*end of opname(namex, namey, nameop)*/

putop(namex, namey, name0, op)
int namex[], namey[], name0[];  char op[];
{int pairname[2];
opname(namex, namey, pairname, op);
putitem(pairname[0], pairname[1], name0);
/*    | | |
makenul(pairname);
*/
putitem(namex[0], namex[1], pairname);
putitem(namey[0], namey[1], pairname);
}/*end of putop(namex, namey, name0)*/

dmp(page, line)  int page, line;
{int j, x[4], y[4];
if (page < 0) return;
setpage(page);  getname(line, x, y);
if(x[0] >= 0)
  {printf(" %d,%d  %d,%d ", page, line, x[0], y[0]);
  prstrng(x[0], y[0]);
  j = 1;  while (++j < 4) if (x[j] >= 0)
  dmp(x[j], y[j]);}
}/*end of dmp(page, line)*/

dmpchfl()
{int j, page, line;
```

```
  j = hlm * hlmline;  while(j--)
    {page = j/hlmline;  line = j%hlmline;
    if (page >= 0) dmp(page, line);}
  }/*end of dmpchfl()*/

  flushbuf()
  {flushch();
  flushln();
  }/*end of flushbuf()*/

  int getay()
  {printf(" (y or n) ");
  flushbuf();
  return(geta('y'));
  }/*end of getay()*/

  int putitem(lp, ll, name)  int lp, ll, name[];
  {int k, tot, j, lpage, lline,
  pl[2], x[4], y[4];
  lpage = name[0];  lline = name[1];
  setpage(lpage);  getlink(lline, x, y);
  /*see if a list has been started yet*/
  if (x[1] < 0)
        {newlink(pl);
        setlink(pl[1], lp, ll, -1, -1, -1, -1, -1, -1);
        setpage(lpage);
        setlink(lline, -2, -2, pl[0], pl[1], -2, -2, -2, -2);
        return(0);}   /*returns 0 if item was absent*/
  tot = j = 0;
  k = 1;  while (1)
        {lpage = x[k];  lline = y[k];
        setpage(lpage);  getlink(lline, x, y);
        newk; /*newk is 'defined' above*/
        /*for debug do*/
        if (debug02)
              {printf(
              "\nin putitem %d %d  %d %d  %d %d  %d %d  %d %d  %d %d  %d %d",
              lp, ll, k, tot, lpage, lline,
              x[0], y[0], x[1], y[1], x[2], y[2], x[3], y[3]);  getcr;}
        /*end debug*/
        /*for debug do*/
        if (x[0] < 0) error("x[0] < 0 in putitem");
        /*end debug*/
        if (x[0] == lp && ll == y[0]) return(1);
        /*returns 1 if item was already present*/
        /*see if the links terminate here*/
        if (x[k] < 0)
              {newlink(pl);
              setlink(pl[1], lp, ll, -1, -1, -1, -1, -1, -1);
              setpage(lpage);
              x[k] = pl[0];  y[k] = pl[1];
              setlink(lline, -2, -2, x[1], y[1], x[2], y[2], x[3], y[3]);
              return(0);}}  /*returns 0 if item was absent*/
  }/*end of putitem(lp, ll, name)*/

  int onlist(lp, ll, name)  int lp, ll, name[];
```

```
(int k, tot, j, lpage, lline, x[4], y[4];
lpage = name[0];   lline = name[1];
setpage(lpage);  getlink(lline, x, y);
if (x[1] < 0) return(0);  /*list is empty*/
tot = j = 0;
k = 1;  while(1)
        (lpage = x[k];   lline = y[k];
        setpage(lpage);  getlink(lline, x, y);
        newk;  /*newk is 'defined' above*/
        /*for debug do*/
        if (debug02)
                (printf(
                "\nin onlist %d %d  %d %d  %d %d  %d %d  %d %d  %d %d  %d %d",
                ip, il, k, tot, lpage, lline,
                x[0], y[0], x[1], y[1], x[2], y[2], x[3], y[3]);  getcr;)
        /*end debug*/
        /*for debug do*/
        if (x[0] < 0) error("x[0] < 0 in onlist");
        /*end debug*/
        if (x[0] == ip && il == y[0]) return(1);
        /*return 1 if item is found */
        /*see if links terminate here */
        if (x[k] < 0) return(0);)
/*return 1 if item is on list, 0 if not on list*/
)/*end of onlist(ip, il, name)*/
```

```
#define lmax 128
#define getcr while (getchar() !=012)


/*      set.c

/*      set.c contains a collection of procedures for
/*      set manipulation

*/


extern setpage();       extern hash01();        extern getname();
extern getlink();       extern newlink();       extern setlink();
extern error();         extern prstrng();       extern reclaim();
extern fetchst();       extern flushbuf();      extern xgetsy();
extern printab();       extern putitem();       extern onlist();

char tlistch[8] {7,6,5,4,3,2,1,0};
int debug03;

pdlist(j, plist, llist, nameb, namec)
int j, plist, llist, nameb[], namec[];
{int l, x[4], y[4];
setpage(plist);  getlink(llist, x, y);
if ((j + onlist(x[0], y[0], nameb))%2)
putitem(x[0], y[0], namec);
l = 4;  while (--l) if (x[l] >= 0)
pdlist(j, x[l], y[l], nameb, namec);
}/*end of pdlist(j, plist, llist, nameb, namec)*/

addlist (plist, llist, name)  int plist, llist, name[];
{int l, x[4], y[4];
if (plist < 0) return;  /*list is empty*/
setpage(plist);  getlink(llist, x, y);
/*for debug do*/
if (x[0] < 0) error("x[0] < 0 in addlist");
/*end debug*/
putitem(x[0], y[0], name);
l = 4;  while (--l) if (x[l] >=0)
addlist(x[l], y[l], name);
}/*end of addlist(plist, llist, name)*/

int sublist(plist, llist, name)  int plist, llist, name[];
{int l, x[4], y[4];
if (plist < 0) return(1);
/*list is empty*/
setpage(plist);  getlink(llist, x, y);
/*for debug do*/
if (x[0] < 0) error("x[0] < 0 in sublist");
/*end debug*/
if (!onlist(x[0], y[0], name)) return(0);
l = 4;  while(--l) if (x[l] >= 0)
if(!sublist(x[l], y[l], name)) return(0);
return(1);
/*sublist need not be proper sublist*/
```

```
}/*end of sublist(plist, llist, name)*/

addset(namea, nameb)  int namea[], nameb[];
{int x[4], y[4];
if (namea[0] == nameb[0] && namea[1] == nameb[1])
return;  /* namea = nameb*/
setpage(namea[0]);  getlink(namea[1], x, y);
if (x[1] < 0) return;  /*association list of namea is empty*/
addlist(x[1], y[1], nameb);
}/*end of addset(namea, nameb)*/

int subset(namea, nameb) int namea[], nameb[];
{int x[4], y[4];
setpage(namea[0]);  getlink(namea[1], x, y);
return(sublist(x[1], y[1], nameb));
/*subset need not be a proper subset*/
}/*end subset(namea, nameb)*/

int seteq(namea, nameb) int namea[], nameb[];
{return(subset(namea, nameb) && subset(nameb, namea));
}/*end of seteq(namea, nameb)*/

prodset(namea, nameb, namec)
int namea[], nameb[], namec[];
{int x[4], y[4];
if (namea[0] == namec[0] && namea[1] == namec[1]
|| nameb[0] == namec[0] && nameb[1] == namec[1])
error("namea = namec or nameb = namec in prodset");
setpage(nameb[0]);  getlink(nameb[1], x, y);
if (x[1] < 0) return; /* association list of nameb is empty*/
setpage(namea[0]);  getlink(namea[1], x, y);
if (x[1] < 0) return;  /*association list of namea is empty*/
pdlist(0, x[1], y[1], nameb, namec);
}/*end of prodset(namea, nameb, namec)*/

difset(namea, nameb, namec)
int namea[], nameb[], namec[];
{int x[4], y[4];
if (namea[0] == namec[0] && namea[1] == namec[1]
|| nameb[0] == namec[0] && nameb[1] == namec[1])
error("namea = namec or nameb = namec in difset");
setpage(namea[0]);  getlink(namea[1], x, y);
if (x[1] < 0) return;  /*association list of namea is empty*/
pdlist(1, x[1], y[1], nameb, namec);
}/*end of dif set(namea, nameb, namec)*/

garbcol(pg, ln)  int pg, ln;
{int j, x[4], y[4];
setpage(pg);  getlink(ln, x, y);
/*for debug do*/
if (debug03) printf("  %d %d ", pg, ln);
/*end debug*/
reclaim(pg, ln);
j = 4;  while (--j) if (x[j] >= 0)
garbcol(x[j], y[j]);
}/*end of garbcol(pg, ln)*/
```

```
makenul(name)   int name[];
{int x[4], y[4];
setpage(name[0]);  getlink(name[1], x, y);
if (x[1] < 0) return;  /*association list of name is already nul*/
setlink(name[1], -2, -2, -1, -1, -2, -2, -2, -2);
garbcol(x[1], y[1]);
}/*end of makenul(name)*/

int printst(p, j)  int p, j;
{int l;
l = prstrng(p, j);
if (l > 15)
        {printf("  continue listing?");  return(getay());}
return(l);
}/*end of printst(p, j)*/

int prnlist(zerol, pg, ln)  int zerol, pg, ln;
{int j, x[4], y[4], xn[4], yn[4];
setpage(pg);  getlink(ln, x, y);
setpage(x[0]);  getname(y[0], xn, yn);
/*for debug do*/
if (xn[0] < 0) error("xn[0] < 0 in prnlist");
/*end debug*/
if (!printst(xn[0], yn[0])) return(0);
if (zerol) return(1);
j = 4;  while (--j) if  (x[j] >= 0)
if (!prnlist(zerol, x[j], y[j])) return(0);
return(1);
}/*end of prnlist(zerol, pg, ln)*/

prinset(zerol, name)  int zerol, name[];
{int x[4], y[4];
setpage(name[0]);  getname(name[1], x, y);
/*for debug do*/
if (debug03)
        {printf("\n in prinset %d %d %d %d ",
        name[0], name[1], x[0], y[0]);  getcr;}
/*end debug*/
/*for debug do*/
if (x[0] < 0) error("x[0] < 0 in prinset");
/*end debug*/
if (!printst(x[0], y[0])) return;
if (x[1] >= 0) prnlist(zerol, x[1], y[1]);
}/*end of prinset(zerol, name)*/

int dsplstn(zerol, one0, page, line, n)  int zerol, one0, page, line, n;
{int j, pl[2], x[4], y[4];
if (page < 0) return(1);
if (n <= 1)
        return(prnlist(zerol, page, line));
setpage(page);  getlink(line, x, y);
pl[0] = x[0];  pl[1] = y[0];
if (!dsplnary(zerol, one0, n-1, pl)) return(0);
j = 4;  while (--j) if (x[j] >= 0)
if(!dsplstn(zerol, one0, x[j], y[j], n)) return(0);
```

```
return(1);
}/*end of dsplstn(zerol, one0, page, line, n)*/

int dsplnary(zerol, one0, n, name)  int zerol, one0, n, name[];
{int x[4], y[4];
setpage(name[0]);   getname(name[1], x, y);
/*for debug do*/
if (x[0] < 0) error(" x[0] < 0 in dsplnary of set.c ");
/*end debug*/
if (one0) {printab(n);  if (!printst(x[0], y[0])) return(0);}
if (x[1] < 0) return(1);
if (n) return(dsplstn(zerol, one0, x[1], y[1], n));
return(1);
}/*end of dsplnary(zerol, one0, n, name)*/

int cntlist(page, line)  int page, line;
{int x[4], y[4];
if (page < 0) return(0);
setpage(page);  getlink(line, x, y);
return(1 +
cntlist(x[1], y[1]) +
cntlist(x[2], y[2]) +
cntlist(x[3], y[3]));
}/*end of cntlist(page, line)*/

int cntset(name)  int name[];
{int x[4], y[4];
setpage(name[0]);  getlink(name[1], x, y);
return(cntlist(x[1], y[1]));
}/*end of cntset(name)*/

exlists(namea, nameb) int namea[], nameb[];
{int xa[4], ya[4], xb[4], yb[4];
setpage(namea[0]);  getlink(namea[1], xa, ya);
setpage(nameb[0]);  getlink(nameb[1], xb, yb);
setlink(nameb[1], -2, -2, xa[1], ya[1], -2, -2, -2, -2);
setpage(namea[0]);
setlink(namea[1], -2, -2, xb[1], yb[1], -2, -2, -2, -2);
/*interchanges lists of namea & nameb*/
}/*end of exlists(namea, nameb)*/

templist(j, name)  int j, name[];
{if (j > 127) error("j > 127 in templist (of set.c)");
tlistch[8-2] = j;
hash01(tlistch, name);
makenul(name);
/*creates a "new" name (for internal use only)
j can run up to 127*/
}/*end of templist(j, name)*/

onallsl(page, line, name)  int page, line, name[];
/*places name on all sublists starting at page, line*/
{int x[4], y[4], pl[2], j;
if (page >= 0)
        {setpage(page);  getlink(line, x, y);
        pl[0] = x[0];  pl[1] = y[0];
```

```
        putitem(name[0], name[1], pl);
        j = 4;  while (--j) if (x[j] >= 0)
        onallsl(x[j], y[j], name);}
}/*end of onallsl(page, line, name);*/

allsub(name1, name2)  int name1[], name2[];
{int x[4], y[4];
setpage(name1[0]);  getlink(name1[1], x, y);
if (x[1] >= 0) onallsl(x[1], y[1], name2);
}/*end of allsub(name1, name2)*/

inlist2(page, line, name2, name3)  int page, line, name2[], name3[];
{int x[4], y[4], pl[2], j;
if (page >= 0)
        {setpage(page);  getlink(line, x, y);
        pl[0] =  x[0];  pl[1] = y[0];
        makenul(name3);
        prodset(pl, name2, name3);
        exlists(name2, name3);
        j = 4;  while (--j) if (x[j] >= 0)
        inlist2(x[j], y[j], name2, name3);}
}/*end of inlist2(page, line, name2, name3)*/

unlist2(page, line, name)  int page, line, name[];
{int x[4], y[4], pl[2], j;
if (page >= 0)
        {setpage(page);  getlink(line, x, y);
        pl[0] = x[0];  pl[1] = y[0];
        addset(pl, name);
        j = 4;  while (--j) if (x[j] >= 0)
        unlist2(x[j], y[j], name);}
}/*end of unlist2(page, line, name)*/

inter2(name1, name2, name3) int name1[], name2[], name3[];
{int x1[4], y1[4], x2[4], y2[4], pl[2];
/*get secondary lists of name1 and intersect (into name2)
using name3 as work space*/
makenul(name2);
setpage(name1[0]);  getlink(name1[1], x1, y1);
if (x1[1] < 0) return;  /*list is empty*/
setpage(x1[1]);  getlink(y1[1], x2, y2);
pl[0] = x2[0];  pl[1] = y2[0];
addset(pl, name2);
inlist2(x1[1], y1[1], name2, name3);
}/*end of inter2(name1, name2, name3)*/

differ2(name1, name2, name3)
int name1[], name2[], name3[];
{int j, x[4], y[4], pl[2];
/*get secondary lists of name1 and 'difference' into name2,
using name3 as work space*/
makenul(name2);
makenul(name3);
setpage(name1[0]);  getlink(name1[1], x, y);
if (x[1] < 0) return;
setpage(x[1]);  getlink(y[1], x, y);
```

```
pl[0] = x[0];  pl[1] = y[0];
j = 4;  while (--j) if (x[j] >= 0)
unlist2(x[j], y[j], name3);
difset(pl, name3, name2);
}/*end of differ2(name1, name2, name3)*/

union2(name1, name2)  int name1[], name2[];
{int x[4], y[4];
/*put (union) all secondary lists of name1 in name2*/
makenul(name2);
setpage(name1[0]);  getlink(name1[1], x, y);
unlist2(x[1], y[1], name2);
}/*end of union2(name1, name2)*/

zfield1(z, str, page, line, name)  int page, line, name[];
char z[], str[];
{int j, c, x[4], y[4], pl[2];
/*adds all z fields of list (page & line) to name*/
if (page < 0) return;  /*list is empty*/
setpage(page);  getlink(line, x, y);
pl[0] = x[0];  pl[1] = y[0];
fetchst(pl, str, lmax);
j = 0;  while ((c = z[j]) == str[j] && c) j =+ 1;
if (!c) putitem(pl[0], pl[1], name);
j = 4;  while (--j) if (x[j] >= 0)
zfield1(z, str, x[j], y[j], name);
}/*end of zfield1(z, str, page, line, name)*/

zfldset(z, name1, name2)
char z[];  int name1[], name2[];
{int x[4], y[4];  char str[lmax];
/*adds all z fields of name1 to name2*/
setpage(name1[0]);  getlink(name1[1], x, y);
if (x[1] >= 0) zfield1(z, str, x[1], y[1], name2);
}/*end of zfldset(z, name1, name2)*/

chngz1(page, line, name, fldstr)
int page, line, name[];  char fldstr[];
{int j, x[4], y[4], pl[2];
if (page < 0) return;  /*list is empty*/
setpage(page);  getlink(line, x, y);
pl[0] = x[0];  pl[1] = y[0];
if (xgetsy(pl, fldstr, pl))
putitem(pl[0], pl[1], name);
j = 4;  while (--j) if (x[j] >= 0)
chngz1(x[j], y[j], name, fldstr);
}/*end of chngz1(page, line, name, fldstr)*/

chngz(name1, name2, fldstr)  int name1[], name2[];  char fldstr[];
{int x[4], y[4];
/*changes the fields of all list elements to field of fldstr*/
makenul(name2);
setpage(name1[0]);  getlink(name1[1], x, y);
if (x[1] >= 0) chngz1(x[1], y[1], name2, fldstr);
}/*end of chngz(name1, name2, fldstr)*/
```

```
#define stksize 16
#define jpathmax 64
#define templim 10
#define getcr while (getchar() != 012)
#define get012 while (chlast != 012) chlast = getchar()
#define curntop temp[6]
#define topics temp[1]
#define all   temp[2]
#define tempa  temp[3]
#define tempb  temp[4]
#define bibmods  temp[5]
#define ppnt temp[0]



/*      user.c

/*      user.c contains procedures and a main for user dialog


/*      needs: u.o lp.o cf.o hl.o set.o

*/


extern differ2();
extern dsplnary();      extern templist();
extern prodset();       extern concat();        extern exlists();
extern printab();       extern opname();        extern xgetsy();
extern error();         extern prntstr();       extern getlink();
extern onlist();        extern getname();       extern setpage();
extern getay();         extern addset();        extern getword();
extern getstr();        extern eql();           extern describe();
extern lnkinit();       extern chinit();        extern flushbuf();
extern inter2();        extern hashno();        extern cntset();
extern hash01();        extern difset();        extern readint();
extern makenul();       extern putitem();       extern union2();
extern prinset();       extern zfldset();       extern chngz();

int debug05;
int restrict[2];
int (*restpro)();
int chlast, nopaths;
char str[64];
int stkname[stksize][2];
int pathele[jpathmax][4];
int tname0[2], tname1[2], tname2[2];
char nosuch[]{
"  no such name in file\n"};
int temp[templim][2];
char tempfill[]{".                    "};
char nocapis[]{
"  do not understand, say again . .\n"};
char sh[256];
int pla[2], plb[2];
```

```
char bibbra[]{"BIB{"};
char modbra[]{"MOD{"};
char abra[]{"A{"};
char bbra[]{"B{"};
char abquote[]{"AB"};
char h1quote[] {"H{1}"};
char h3quote[] {"H{3}"};
char h42quote[] {"H{4-2}"};
char dowesave[]{
"  Do you wish to SAVE these items for future reference?"};
char wishsee[]{
"  Do you wish to SEE some of them?"};

int emptypro() {;}

userinit()
{int j;
chinit();
lnkinit();
j = templim;  while (j--)
        {tempfil[j+4] = 0;
        hash01(tempfil, temp[j]);
        makenul(temp[j]);}
templist(0, tname0);  templist(1, tname1);  templist(2, tname2);
j = stksize;  while (j--) templist (j + 6, stkname[j]);
restrict[0] = tname0[0];
restrict[1] = tname0[1];
/*for debug do
printf("  debug05 = ");
flushbuf();
debug05 = readint(0);
/*end debug*/
}/*end of userinit()*/

makerest(bmname)  int bmname[];
{/*restrict[] is global!*/
makenul(tname1);
prodset(restrict, bmname, tname1);
exlists(bmname, tname1);
makenul(tname1);
}/*end of makerest()*/

getrest()
{restpro = emptypro;
printf("  Do you wish a restriction? ");
if (!getay()) return;
printf(" What restriction . . (give as H{4-2} or D{2} or F{1}  etc) \n");
while (!findname(restrict)) printf("  .  . try again\n");
restpro = makerest;
}/*end of getrest()*/

int findname(name)  int name[];
{getstr(str, 64, &chlast);
if (hashno(str, name)) return(1);  else
        {printf(nosuch);  get012;  return(0);}
}/*end of findname(name)*/
```

```
int findword(chs)  char *chs;
{getword(str, 64, &chlast);
if (eql(str, chs)) return(1);  else
        {printf(nocapis);  get012;  return(0);}
}/*end of findword(chs)*/

int geterms(strng, name, tail)  char tail, strng[];  int name[];
{int c, j, jin, pl[2];
makenul(name);
printf("(if more than one separate by TABs)\n");
jin = (-1);  while(sh[++jin] = strng[jin]);
c = 0;  while (c != 012)
        {flushbuf();
        c = 0;
        j = jin;  while (c != 012 && c != 011)
                {j =+ getstr(&sh[j], (256-2) - j, &c);
                if (j < 256-1) j =+ 1;}
        sh[j-1] = tail;
        sh[j] = 0;
        /*for debug do*/
        if (debug05) printf(sh);
        /*end debug*/
        if (!hashno(sh, pl)) hash01(" ", pl);
        putitem(pl[0], pl[1], name);}
return(j);
}/*end of geterms(strng, name)*/

int getbms(strng)  char *strng;
{union2(topics, all);
/*for debug do*/
if (debug05) prinset(0, all);
/*end debug*/
makenul(bibmods);
if (eql(strng, abquote)) addset(all, bibmods);
else zfldset(strng, all, bibmods);
(*restpro)(bibmods);
return(cntset(bibmods));
}/*end of int getbms(strng)*/

advtops(strngx, strngy)  char *strngx, *strngy;
{(*restpro)(all);
union2(all, tempa);
makenul(tempb);
zfldset(strngy, tempa,  tempb);
makenul(tempa);
if (eql(strngx, strngy)) exlists(tempb, tempa);
else chngz(tempb, tempa, strngx);
makenul(topics);
difset(tempa, cumtop, topics);
addset(tempa, cumtop);
makenul(tempb);
makenul(tempa);
/*for debuc do*/
if (debug05) prinset(0, topics);
/*end debug*/
```

```
}/*end of advtops(strnex, strnoy)*/

remitem(rname)  int rname[];
{int c, flag, n, name[2];
printf(
"  by what name do you want these remembered\n");
flag = 1;  while (flag)
        {flushbuf();
        getstr(sh, 256, &c);
        if (hash01(sh, name) && (n = cntset(name)))
                {printf(
"  that name is already in use and has %d items associatedwith it;\n", n);
                printf(
"  are you sure you want to add these items to the list of ");
                printf(sh);
                printf("?");
                if (getay()) flag = 0;
                else printf(
"  give another name for remembering these items\n");}
        else flag = 0;}
addset(rname, name);
printf("  they have been added to ");
printf(sh);  putchar(012);
}/*end of remitem(rname)*/

dsptops(n)  int n;
{printf("  there are %d topics,", n);
printf(wishsee);
if (getay()) prinset(0, topics);
}/*end of dsptops()*/

presbms(strx, stry, strbm)  char *strx, *stry, *strbm;
{int flag, l, n, j;
flag = 1;
l = 0;
while (flag)
        {n = getbms(strbm);
        printf(
        "  there are %d, order %d items:", n, l);
        if (n)  {printf(wishsee);
                if (getay()) dsplnary(1, 1, 2, bibmods);
                printf(dowesave);
                if (getay()) remitem(bibmods);}
        printf(
        "  do you wish to COUNT the order %d items?", ++l);
        if (getay()) advtops(strx, stry);  else flag = 0;
        if (!(j=cntset(topics)))
                {printf("  there are no more associations\n");
                flag = 0;}
        if (flag) dsptops(j);}
}/*end of presbms(strx, stry, strbm)*/

ptopic(strx, stry, strbm)  char *strx, *stry, *strbm;
{int j;
do      {printf(
        "  what is the topic(s) of interest ");
```

```
		makenul(cumtop);
		j = geterms(strx, topics, '}');
		addset(topics, cumtop);
		getrest();
		presbms(strx, strv, strbm);
		printf("  continue this task with other topics?");}
while (getay());
makenul(cumtop);
makenul(topics);
makenul(all);
makenul(bibmods);
}/*end of ptopic(str, stry, strbm)*/

findsubj()
{int j, c, knt, knta, kntb, flag;
printf(
"  Do you wish to inquire about the subject matter in the system?");
if (!getay()) return;
printf("  In what ");
flag = 1;  while(flag)
		{printf(" topic or subtopic are you interested?\n");
		flushbuf();
		j = getstr(&sh[2], 256-4, &c) + 2;
		sh[0] =  'A';
		sh[1] =  '{';
		sh[j] =  '}';
		sh[++j] = 0;
		knta = kntb = 0;
		if (hashno(sh, pla)) knta = cntset(pla);
		sh[0] =  'B';
		if (hashno(sh, plb)) kntb = cntset(plb);
		knt = knta + kntb;
		if (!knt) printf("  Nothing by that name in the system\n");
		else	{printf("  The system has %d entries related to ", knt);
			printf(&sh[1]);
			putchar(012);
			printf(wishsee);
			if (getay())
				{if (knta) dsplnary(1, 1, 2, pla);
				if (kntb) dsplnary(1, 1, 2, plb);}}
		printf("  Do you wish to enquire further about the subject matter?");
		if (!getay()) flag = 0;
		if (flag) printf("  In what other ");}
}/*end of findsubj()*/

int getopics(strx)  char *strx;
{union2(bibmods, tempa);
makenul(topics);
sfldset(strx, tempa, topics);
makenul(tempa);
return(cntset(topics));
}/*end of int getopics(strx)*/

pbibmod(strx, stry, strbm)  char *strx, *stry, *strbm;
{int j, n;
do	{printf(
```

```
            "  what is the item number(s) of interest ");
            makenul(cumtop);
            j = geterms(strbm, bibmods, '} );
            getrest();
            n = getopics(strx);
            addset(topics, cumtop);
            if (n) dsptops(n);
            preshms(strx, stry, strbm);
            printf(
            "  Continue this task with other items?");}
while (getay());
makenul(cumtop);
makenul(topics);
makenul(all);
makenul(bibmods);
}/*end of pbibmod(strx, stry, strbm)*/

pxyiter(x,y)  char x[], y[];
/*note carefully exactly what happens here!*/
/*chngz and union2 will initialize destination to null*/
{if (!eql(x, y))
            {chngz(ppnt, tname0, x);
            exlists(ppnt, tname0);}
union2(ppnt,tname0);
exlists(ppnt, tname0);
(*restpro)(ppnt);
union2(ppnt, tname0);
makenul(ppnt);
ztldset(y, tname0, ppnt);
}/*end of pxyiter(x, y)*/

int pxycon0(x, y, xpl)  char x[], y[];  int xpl[];
{int j;
makenul(tname2);
j = 0;
do          {j =+ 1;
            addset(ppnt, tname2);
            pxyiter(x, y);
            makenul(tname0);
            difset(ppnt, tname2, tname0);
            exlists(ppnt, tname0);
            if (onlist(xpl[0], xpl[1], ppnt))
                        {printf(" connection at level = %d\n", j);
                        return(j);}
            printf(" no connection yet at level = %d ", j);
            printf("count = %d  continue search?", cntset(ppnt));}
while (getay());
makenul(tname2);
return(j);
}/*end of int pxycon0(x, y, xpl)*/

pairput(ja, jb, name, xa, flag)
int flag, ja, jb, name[];  char *xa;
{int plx[2], ply[2], plz[2];
ply[0] = pathele[ja][0];
ply[1] = pathele[ja][1];
```

```
if (!xgetsy(ply, xa, plx))
        {printf(xa);
        error(" in pairput of user.c;  cant find field-value");}
ply[0] = pathele[jb][0];
ply[1] = pathele[jb][1];
opname(plx, ply, plz, "INTER{");
if (flag) {dsplnary(1, 1, 2, plz);  return;}
makenul(plz);
prodset(plx, ply, plz);
putitem(plz[0], plz[1], name);
}/*end of pairput(ja, jb, name, xa, flag)*/

int getjb(j, l)  int j, l;
{int kb, k;
kb = 1 + pathele[j][2];
while (1)
        {k = pathele[l][2];
        if (k == kb) return(l);
        if (k < kb) return(jpathmax);
        if (++l >= jpathmax) return(jpathmax);}
}/*end of int getjb(j, l)*/

procpair(ja, l, name, xa)
int ja, l, name[];  char *xa;
{int jb;
jb = getjb(ja, l);
if (jb >= jpathmax) return;
pairput(ja, jb, name, xa, 0);
if (jb + 1 >= jpathmax) return;
procpair(ja, jb + 1, name, xa);
procpair(jb, jb + 1, name, xa);
}/*end of procpair(ja, l, name, xa)*/

pathpairs(name, xa)  int name[];  char *xa;
{int j0;
j0 = jpathmax;  while (pathele[--j0][2]);
if (j0 >= 0 && j0 + 1 < jpathmax)  procpair(j0, j0 + 1, name, xa);
}/*end of pathpair(name, xa)*/

rempath(page, line, kappa, nops)
int page, line, kappa, nops;
{static int j;  int x[4], y[4];
if (page < 0)
        {j = jpathmax;
        pathele[j-1][2] = 0;
        return;}
if (--j < 0)
        {j = 0;
        printf("\n too many path elements!\n");}
pathele[j][0] = page;
pathele[j][1] = line;
pathele[j][2] = kappa;
pathele[j][3] = nops;
setpage(page);  getname(line, x, y);
printab(kappa);
printf("%d.%d: ", kappa, nops);
```

```
prntstr(x[0], y[0], 0, 0);  putchar(012);
}/*end of rempath(page, line, kappa, nops)*/

int chkitem(page, line, k, xa, ya, xpl, kmax)
int k, page, line, xpl[], kmax;  char xa[], ya[];
{int l;
makenul(stkname[k]);
putitem(page, line, stkname[k]);
exlists(ppnt, stkname[k]);
pxyiter(xa, ya);
exlists(stkname[k], ppnt);
makenul(stkname[k+1]);
l = 0;  while (++l < k)
addset(stkname[l], stkname[k+1]);
makenul(stkname[k+2]);
difset(stkname[k], stkname[k+1], stkname[k+2]);
exlists(stkname[k], stkname[k+2]);
makenul(stkname[k+1]);
makenul(stkname[k+2]);
l = 0;
if (onlist(xpl[0], xpl[1], stkname[k]))
          {l =+ 1;
          nopaths =+ 1;
          rempath(xpl[0], xpl[1], k, nopaths);
          rempath(page, line, k-1, nopaths);}
if (k < kmax)
if (conset(stkname[k], k, xa, ya, xpl, kmax))
          {l =+ 1;
          rempath(page, line, k-1, nopaths);}
return(l);
}/*end of int chkitem(page, line, k, xa, ya, xpl, kmax)*/

int conlist(page, line, k, xa, ya, xpl, kmax)
int page, line, k, xpl[], kmax;  char xa[], ya[];
{int j, l, pl[2], x[4], y[4];
if (page < 0) return(0);
setpage(page);  getlink(line, x, y);
l = chkitem(x[0], y[0], k, xa, ya, xpl, kmax);
j = 4;  while (--j) if (x[j] >= 0)
l =+ conlist(x[j], y[j], k, xa, ya, xpl, kmax);
return(l);
}/*end of int conlist(page, line, k, xa, ya, spl, kmax)*/

int conset(name, k, xa, ya, xpl, kmax)
int k, name[], xpl[], kmax;  char xa[], ya[];
{int x[4], y[4], l;
setpage(name[0]);  getlink(name[1], x, y);
if (x[1] < 0) return(0);
k =+ 1;
l = conlist(x[1], y[1], k, xa, ya, xpl, kmax);
makenul(stkname[k]);
return(l);
}/*end of int conset(name, k, xa, ya, xpl, kmax)*/

pxypath(x, y, kmax, xpl)  char x[], y[];  int kmax, xpl[];
{rempath(-1, 0, 0, 0);
```

```
nopaths = 0;
do      {if (conset(ppnt, 0, x, y, xpl, kmax))
                {makenul(tname0);
                pathpairs(tname0, x);
                exlists(ppnt, tname0);
                printf("        %d  paths \n", nopaths);
                return;}
        printf("no connection yet at level = %d;  continue search?", kmax);
        kmax =+ 1;}
while (getay() && kmax+2 < stksize);
}/*end of pxypath(x, y, kmax, xpl)*/


dispmsg(mess)  char *mess;
{printf(mess);
printf("\n do you want to see another individual path?");
}/*end of dispmsg(mess)*/

int getjokn(ki, ni, j)  int ki, ni, j;
{int kappa, nops;
kappa = 1;  while (kappa)
        {kappa = pathele[j][2];
        nops = pathele[j][3];
        if (kappa == ki && nops == ni) return(j);
        if (--j < 0) return(-1);}
return(-1);
}/*end of getjokn(ki, ni, j)*/

dispind(xa)  char *xa;
{int ki, ni, kf, nf, ja, jb, name[];
printf("  Give junction pair numbers (higher first, then lower). .\n");
ki = readint(0);
ni = readint(0);
kf = readint(0);
nf = readint(0);
if (ni > nf || ni > nopaths || kf >= ki)
        {dispmsg("  Incompatible starting and ending junction pair . .");
        return;}
jb = getjokn(ki, ni, jpathmax);
if (jb < 0)
        {printf("%d.%d  ", ki, ni);
        dispmsg("no corresponding junction . .");
        return;}
while (1)
        {ki =- 1;
        if (ki < 0)
                {dispmsg("cant find junction . .");
                return;}
        if (ki == kf) ni = nf;
        ja = (-1);  while (ja < 0)
                {ja = getjokn(ki, ni, jb);
                if (ja < 0) ni =+ 1;
                if (ni > nopaths || ni > nf)
                        {dispmsg("no next junction . .");
                        return;}}
        pairput(ja, jb, name, xa, 1);
```

```
        if (kf >= ki || ni > nf)
                {dispmsg(" ");  return;}
        jb = ja;}
}/*end of dispind(xa)*/

findnm(xas, xa, name)  int name[]; char xas[], *xa;
{getstr(str, 64, &chlast);
xas[0] = 0;
concat(xas, xa, 64);
concat(xas, str, 64);
concat(xas, "}", 64);
if (xas[64-2]) xas[64-2] = '}';
xas[64-1] = 0;
/*for debug do*/
if (debug05) printf(xas);
/*end debug*/
if (hashno(xas, name)) return(1);
else    {printf(nosuch);  get012;  return(0);}
}/*end of findnm(xas, xa, name)*/

pconpath(xa, ya)  char *xa, *ya;
{char xas[64], yas[64], nm[128], *ch;
int starterm[2], endterm[2], j, name[2];
printf("give starting term:  ");  flushbuf();
if (!findnm(xas, xa, starterm)) return;
/*note the starting term begins as xa!*/
printf("give ending term for path:  ");  flushbuf();
if (!findnm(yas, ya, endterm)) return;
getrest();
makenul(ppnt);  putitem(starterm[0], starterm[1], ppnt);
j = pxycon((xa, ya, endterm);
printf(
"  There are %d terms at level %d;  do you wish to see them?",
cntset(ppnt), j);
if (getay()) prinset(0, ppnt);
printf(
"  Do you wish to construct paths?");
if (!getay()) return;
makenul(ppnt);  putitem(starterm[0], starterm[1], ppnt);
pxypath(xa, ya, j, endterm);
printf(
"  Do you wish to see ALL the items (modules or bibliographic entries) ");
printf("\nalong ALL the paths?");
if (getay()) dsplnary(1, 1, 3, ppnt);
printf("  Do you wish to examine individual paths?");
while (getay()) dispind(xa);
printf(
"  Do you wish this path information remembered?");
if (!getay()) return;
nm[0] = 0;
concat(nm, "PATHS{", 128);
concat(nm, xas, 128);
concat(nm, ":", 128);
concat(nm, yas, 128);
concat(nm, "}", 128);
if (nm[128-2]) nm[128-2] = '}';
```

```
nm[128-1] = 0;
/*for debug do*/
if (debug05) printf(nm);
/*end debug*/
hash01(nm, name);
addset(ppnt, name);
printf(
"the paths will be remembered under the name \n");
printf(nm);
makenul(ppnt);
}/*end of pconpath(xa, ya)*/

puid(strl, str2)  char *strl, *str2;
{int j, n, flag, (*func)();
do      {do     {flag = 0;
                 printf("--union, intersection, or difference?.. ");
                 getword(str, 64, &chlast);
                 str[3] = 0;
                 if (eql(str, "uni")) func = union2; else
                 if (eql(str, "int")) func = inter2; else
                 if (eql(str, "dif")) func = differ2; else
                        {printf(nocapis);  get012;  flag = 1;}}
        while (flag);
        printf(
        "  What are the fields and values of interest?\n");
        printf(
        "  Present as A{..}, D{..}, C{.., etc  ");
        j = geterms("\0", topics, 0);
        makenul(bibmods);
        (*func)(topics, all, tempa);
        if (eql(strl, str2))
        zfldset(strl, all, bibmods);
        else exists(all, bibmods);
        n = cntset(bibmods);
        printf(
        "there are %d items, ", n);
        if (n)  {printf(wishsee);
                 if (getay()) dspinary(1, 1, 2, bibmods);
                 printf(dowesave);
                 if (getay()) remitem(bibmods);}
        printf(
        "..continue this task with other operations, fields and values?");}
while (getay());
makenul(tempa);
makenul(topics);
makenul(all);
makenul(bibmods);
}/*end of puid(strl, str2)*/

ptaskhx(hfield, xa)  char *hfield, *xa;
{char xas[64];  int j, namex[2], nameh[2];
hash01(hfield, nameh);
do      {j = 1;  while (j)
                {printf("  Give the subtopic or topic of interest: ");
                makenul(ppnt);
                flushbuf();
```

```
                if (j = !findnm(xas, xa, namex))
                printf("  There aren't any;");}
        prodset(namex, nameh, ppnt);
        printf("  There are %d items in this catagory\n",
        j = cntset(ppnt));
        if (j)  {printf(wishsee);
                if (getay()) dsplnary(1, 1, 2, ppnt);
                printf(dowesave);
                if (getay()) remitem (ppnt);}
        printf("  Continue this task with another topic?");}
while(getay());
makenul(ppnt);
}/*end of ptaskhx(hfield, xa)*/

findtask()
{int flag, c;
flushbuf();
while (describe("tchoice.alf"));
flag = 1;
while (flag)
        {printf("  which task do you now wish.. ");
        flushbuf();
        restpro = emptypro;
        getstr(sh, 256, &c);
        switch(sh[0]) {
case '9':       if (eql(sh, "9"))        pbibmod(bbra, abra, bibbra); else
case '8':       if (eql(sh, "8"))        pbibmod(abra, bbra, modbra); else
case '7':       if (eql(sh, "7"))        pbibmod(abra, bbra, bibbra); else
case '6':       if (eql(sh, "6"))        ptopic(bbra, bbra, modbra); else
                if (eql(sh, "6&5"))      ptopic(bbra, bbra, abquote); else
case '5':       if (eql(sh, "5"))        ptopic(bbra, bbra, bibbra); else
                if (eql(sh, "5&6"))      ptopic(bbra, bbra, abquote); else
case '4':       if (eql(sh, "4"))        ptopic(bbra, abra, modbra); else
                if (eql(sh, "4&3"))      ptopic(bbra, abra, abquote); else
case '3':       if (eql(sh, "3"))        ptopic(bbra, abra, bibbra); else
                if (eql(sh, "3&4"))      ptopic(bbra, abra, abquote); else
case '2':       if (eql(sh, "2"))        ptopic(abra, bbra, modbra); else
                if (eql(sh, "2&1"))      ptopic(abra, bbra, abquote); else
case '1':       if (eql(sh, "1"))        ptopic(abra, bbra, bibbra); else
                if (eql(sh, "1&2"))      ptopic(abra, bbra, abquote); else
                if (eql(sh, "18"))       puid(modbra, modbra); else
                if (eql(sh, "18&17"))    puid(bibbra, modbra); else
                if (eql(sh, "17"))       puid(bibbra, bibbra); else
                if (eql(sh, "17&18"))    puid(bibbra, modbra); else
                if (eql(sh, "15"))       pconpath(bbra, bbra); else
                if (eql(sh, "14"))       pconpath(bbra, abra); else
                if (eql(sh, "13"))       pconpath(abra, bbra); else
                if (eql(sh, "12"))       pbibmod(bbra, bbra, modbra); else
                if (eql(sh, "11"))       pbibmod(bbra, bbra, bibbra); else
                if (eql(sh, "10"))       pbibmod(bbra, abra, modbra); else
case '0':       if (eql(sh, "0"))        flag = 0;       else
case 'h':       if (eql(sh, "halt"))     flag = 0;       else
                if (eql(sh, "help"))     while(describe("tchoice.alf")); else
case 'T':       if (eql(sh, "T-5"))      ptopic(abra, bbra, bibbra); else
                if (eql(sh, "T-5-exp"))  ptopic(abra, bbra, bibbra); else
                if (eql(sh, "T-5-rex"))  ptopic(bbra, abra, bibbra); else
```

```
                    if (eql(sh, "T-5-con"))  ptopic(bbra, bbra, bibbra); else
                    if (eql(sh, "T-14"))     pbibmod(abra, bbra, bibbra); else
                    if (eql(sh, "T-14-exp")) pbibmod(abra, bbra, bibbra); else
                    if (eql(sh, "T-14-rex")) pbibmod(bbra, abra, bibbra); else
                    if (eql(sh, "T-14-con")) pbibmod(bbra, bbra, bibbra); else
                    if (eql(sh, "T-2"))      ptaskhx(h3quote, abra); else
                    if (eql(sh, "T-2-hom"))  ptaskhx(h3quote, abra); else
                    if (eql(sh, "T-2-con"))  ptaskhx(h3quote, bbra); else
                    if (eql(sh, "T-4"))      ptaskhx(h42quote, abra); else
                    if (eql(sh, "T-4-hom"))  ptaskhx(h42quote, abra); else
                    if (eql(sh, "T-4-con"))  ptaskhx(h42quote, bbra); else
                    if (eql(sh, "T-15"))     ptaskhx(hlquote, abra); else
                    if (eql(sh, "T-15-hom"))        ptaskhx(hlquote, abra); else
                    if (eql(sh, "T-15-con"))        ptaskhx(hlquote, bbra); else
default:            {flag = 1;  printf(nocapis);}
        }           }
}/*end of findtask()*/

task()
{flushbuf();
while (describe("subjs.alf"));
findsubj();
flushbuf();
while (describe("tasks.alf"));
findtask();
}/*end of task()*/

main()
{while(describe("intro.alf"));
userinit();
task();
printf(
"\nThe %c indicates you have returned to the UNIX operating system:\n", '%');
}/*end of main()*/
```

```
#define stksize 16
#define jpathmax 64
#define get012 while (chlast != 012) chlast = getchar()



/*      inter.c

/*      inter.c contains proceedures and a main of the
/*      interactive manipulation of the association lists

/*      needs the following files:  u.o lp.o cf.o hl.o set.o

*/



extern printab();
extern opname();        extern xgetsy();        extern error();
extern chngzl();        extern prntstr();       extern chngz();
extern hashno();        extern geta();
extern describe();      extern getstr();        extern getword();
extern union2();        extern eol();           extern inter2();
extern dsplnary();      extern zfldset();       extern getlink();
extern difset();        extern putop();         extern exlists();
extern allsub();        extern onlist();        extern addset();
extern flushbuf();      extern chinit();        extern lnkinit();
extern templist();      extern getname();       extern setpage();
extern printst();       extern prinset();       extern prodset();
extern cntset();        extern hash01();        extern subset();
extern makenul();       extern readint();       extern putitem();
int chlast, nopaths, flgp, ppnt[2], oldtim0, oldtim1, timex[6];
int tname0[2], tname1[2], tname2[2];
char nocapis[] {
"  do not understand, say again . . \n"};
char nosuch[]{
"  no such name in file\n"};
char str[64];
int stkname[stksize][2];
int pathele[jpathmax][3];

intinit()
{int k;
chinit();
lnkinit();

printf(
"\nThis is the interactive manipulator for association lists.");
while (describe("inter.alf"));
printf("Make timing measurements? (y or n) ");
if (geta('y')) oldtim1 = 1; else oldtim1 = 0;
printf("enter requests:\n>");
templist(0, tname0);  templist(1, tname1);  templist(2, tname2);
k = stksize;  while (k--) templist (k + 6, stkname[k]);
flushbuf();
```

```
getword(str, 64, &chlast);
flgp = 1;
}/*end of intinit()*/

int findname(name)  int name[];
{getstr(str, 64, &chlast);
if (hashno(str, name)) return(1);  else
        {printf(nosuch);  get012;  return(0);}
}/*end of findname(name)*/

int findword(chs)  char *chs;
{getword(str, 64, &chlast);
if (eql(str, chs)) return(1);  else
        {printf(nocapis);  get012;  return(0);}
}/*end of findword(chs)*/

pcreate()
{int p1[2];
getstr(str, 64, &chlast);
if (hash01(str, p1))
        {printf("\n that name already exists;  may not create it\n");  return;}
flgp = 0;
ppnt[0] = p1[0];
ppnt[1] = p1[1];
}/*end of pcreate()*/

ppoint()
{int p1[2];
getstr(str, 64, &chlast);
if (!hashno(str, p1))
        {printf("\nthat name does not yet exist in file;  can not point to it\n")
        return;}
flgp = 0;
ppnt[0] = p1[0];
ppnt[1] = p1[1];
}/*end of ppoint()*/

pcount()
{printf("  %d \n", cntset(ppnt));
}/*end of pcount()*/

pdisplay()
{int x[4], y[4];
getword(str, 64, &chlast);
if (eql(str, "secondary"))
        {dsplnary(0, 1, 2, ppnt);  return;}
if (eql(str, "name"))
        {setpage(ppnt[0]);  getname(ppnt[1], x, y);
        printst(x[0], y[0]);  putchar(012);  return;}
if (eql(str, "n-ary"))
        {dsplnary(readint(0), 1, readint(0), ppnt);  return;}
if(!eql(str, "set"))
        {printf(nocapis);  get012;  return;}
prinset(0, ppnt);
}/*end of pdisplay()*/
```

```
pattach()
{int xpl[2];
if (!findname(xpl)) return;
putitem(xpl[0], xpl[1], ppnt);
}/*end of pattach()*/

pis()
{int xpl[2];
if (!findword("member"))
return;
if (!findname(xpl)) return;
if (cnlist(xpl[0], xpl[1], ppnt))
printf("  yes\n");  else printf("  no\n");
}/*end of pis()*/

prelation()
{int xpl[2], sub, sup, j;
if (!findword("set"))
return;
if (!findname(xpl)) return;
sub = subset(xpl, ppnt);
sup = subset(ppnt, xpl);
if (sub && sup)
        {printf("  identical\n");  return;}
if (sub) {printf("  subset\n");  return;}
if(sup) {printf("  superset\n");  return;}
makenul(tname0);
prodset(xpl, ppnt, tname0);
j = cntset(tname0);
if (j) printf("  %d \n", j);
else printf("  disjoint\n");
}/*end of prelation()*/

padd()
{int xpl[2];
if (!findword("set"))
return;
if (!findname(xpl)) return;
addset(xpl, ppnt);
}/*end of padd()*/

pdelete()
{int xpl[2], flg;
getword(str, 64, &chlast);
if (eql(str, "set")) flg = 1; else
if (eql(str, "setnot")) flg = 0; else
        {printf(nocapis);  get012;  return;}
if (!findname(xpl)) return;
makenul(tname0);
if (flg) difset(ppnt, xpl, tname0);
else prodset(ppnt, xpl, tname0);
exlists(ppnt, tname0);
}/*end of pdelete()*/

premove()
{int xpl[2];
```

```
if (!findname(xpl)) return;
makenul(tname0);
makenul(tname1);
putitem(xpl[0], xpl[1], tname0);
difset(ppnt, tname0, tname1);
exlists(ppnt, tname1);
makenul(tname1);
}/*end of premove()*/

pmake()
{if (!findword("set"))
return;
if (!findword("null"))
return;
makenul(ppnt);
}/*end of pmake()*/

pon()
{int xpl[2];
if (!findword("secondary"))
return;
if (!findword("put"))
return;
if (!findname(xpl)) return;
makenul(tname0);
addset(ppnt, tname0);
allsub(tname0, xpl);
/*put xpl on all secondary ppnt*/
}/*end of pon()*/

punion()
{int xpl[2];
if (!findword("secondary"))
return;
if (!findname(xpl)) return;
union2(xpl, tname0);
exlists(ppnt, tname0);
}/*end of punion()*/

pinter()
{int xpl[2];
if (!findword("secondary"))
return;
if (!findname(xpl)) return;
inter2(xpl, tname0, tname1);
exlists(ppnt, tname0);
makenul(tname1);
}/*end of pinter()*/

pxyiter(x,y)  char x[], y[];
/*note carefully exactly what happens here!*/
/*chngz and union2 will initialize destination to null*/
{if (!eql(x, y))
        {chngz(ppnt, tname0, x);
        exlists(ppnt, tname0);}
union2(ppnt,tname0);
```

```
exlists(ppnt, tname0);
union2(ppnt, tname0);
makenul(ppnt);
zfldset(y, tname0, ppnt);
}/*end of pxyiter(x, y)*/

pxycon0(x, y)  char x[], y[];
{int j, xpl[2];
if (!findword("to"))
return;
if (!findname(xpl)) return;
makenul(tname2);
j = 0;
do      {j =+ 1;
        addset(ppnt, tname2);
        pxyiter(x, y);
        makenul(tname0);
        difset(ppnt, tname2, tname0);
        exlists(ppnt, tname0);
        if (onlist(xpl[0], xpl[1], ppnt))
                {printf(" connection at level = %d\n", j);
                return;}
        printf(" no connection yet at level = %d ", j);
        printf("count = %d  continue search? (y or n) ", cntset(ppnt));
        flushbuf();}
while (geta('y'));
makenul(tname2);
}/*end of pxycon0(x, y)*/

pairput(ja, jb, name, xa)
int ja, jb, name[];  char *xa;
{int plx[2], ply[2], plz[2];
ply[0] = pathele[ja][0];
ply[1] = pathele[ja][1];
if (!xgetsy(ply, xa, plx))
        {printf(xa);
        error(" in pairput of inter.c;  cant find field-value");}
ply[0] = pathele[jb][0];
ply[1] = pathele[jb][1];
opname(plx, ply, plz, "INTER{");
makenul(plz);
prodset(plx, ply, plz);
putitem(plz[0], plz[1], name);
}/*end of pairput(ja, jb, name, xa)*/

int getjb(j, l)  int j, l;
{int kb, k;
kb = 1 + pathele[j][2];
while (1)
        {k = pathele[l][2];
        if (k == kb) return(l);
        if (k < kb) return(jpathmax);
        if (++l >= jpathmax) return(jpathmax);}
}/*end of int getjb(j, l)*/

procpair(ja, l, name, xa)
```

```
int ja, l, name[];  char *xa;
{int jb;
jb = getjb(ja, l);
if (jb >= jpathmax) return;
pairput(ja, jb, name, xa);
if (jb + 1 >= jpathmax) return;
procpair(ja, jb + 1, name, xa);
procpair(jb, jb + 1, name, xa);
}/*end of procpair(ja, l, name, xa)*/

pathpairs(name, xa)  int name[];  char *xa;
{int j0;
j0 = jpathmax;  while (pathele[--j0][2]);
if (j0 >= 0 && j0 + 1 < jpathmax)  procpair(j0, j0 + 1, name, xa);
}/*end of pathpair(name, xa)*/

rempath(page, line, kappa)
int page, line, kappa;
{static int j;  int x[4], y[4];
if (page < 0)
        {j = jpathmax;
        pathele[j-1][2] = 0;
        return;}
if (--j < 0)
        {j = 0;
        printf("\n too many path elements!\n");}
pathele[j][0] = page;
pathele[j][1] = line;
pathele[j][2] = kappa;
setpage(page);  getname(line, x, y);
printab(kappa);  printf("%d: ", kappa);
prntstr(x[0], y[0], 0, 0);  putchar(012);
}/*end of rempath(page, line, kappa)*/

int chkitem(page, line, k, xa, ya, xpl, kmax)
int k, page, line, xpl[], kmax;  char xa[], ya[];
{int l;
makenul(stkname[k]);
putitem(page, line, stkname[k]);
exlists(ppnt, stkname[k]);
pxyiter(xa, ya);
exlists(stkname[k], ppnt);
makenul(stkname[k+1]);
l = 0;  while (++l < k)
addset(stkname[l], stkname[k+1]);
makenul(stkname[k+2]);
difset(stkname[k], stkname[k+1], stkname[k+2]);
exlists(stkname[k], stkname[k+2]);
makenul(stkname[k+1]);
makenul(stkname[k+2]);
l = 0;
if (onlist(xpl[0], xpl[1], stkname[k]))
        {l =+ 1;
        nopaths =+ 1;
        rempath(xpl[0], xpl[1], k);
        rempath(page, line, k-1);}
```

```
if (k < kmax)
if (conset(stkname[k], k, xa, ya, xpl, kmax))
        {l =+ 1;
        rempath(page, line, k-1);}
return(l);
}/*end of int chkitem(page, line, k, xa, ya, xpl, kmax)*/

int conlist(page, line, k, xa, ya, xpl, kmax)
int page, line, k, xpl[], kmax;  char xa[], ya[];
{int j, l, pl[2], x[4], y[4];
if (page < 0) return(0);
setpage(page);  getlink(line, x, y);
l = chkitem(x[0], y[0], k, xa, ya, xpl, kmax);
j = 4;  while (--j) if (x[j] >= 0)
l =+ conlist(x[j], y[j], k, xa, ya, xpl, kmax);
return(l);
}/*end of int conlist(page, line, k, xa, ya, spl, kmax)*/

int conset(name, k, xa, ya, xpl, kmax)
int k, name[], xpl[], kmax;  char xa[], ya[];
{int x[4], y[4], l;
setpage(name[0]);  getlink(name[1], x, y);
if (x[1] < 0) return(0);
k =+ 1;
l = conlist(x[1], y[1], k, xa, ya, xpl, kmax);
makenul(stkname[k]);
return(l);
}/*end of int conset(name, k, xa, ya, xpl, kmax)*/

pxypath(x, y)  char x[], y[];
{int kmax, xpl[2];
if (!findword("to"))
return;
if (!findname(xpl)) return;
printf(  " to what level? . . ");
flushbuf();
kmax = readint(0);
rempath(-1, 0, 0);
nopaths = 0;
do      {if (conset(ppnt, 0, x, y, xpl, kmax))
                {makenul(tname0);
                pathpairs(tname0, x);
                exlists(ppnt, tname0);
                printf("      %d  paths \n", nopaths);
                return;}
        printf("no connection yet at level = %d;  continue search?  ", kmax);
        kmax =+ 1;  flushbuf();}
while (geta('y') && kmax+2 < stksize);
}/*end of pxypath(x, y)*/

rxy2n(namex, namey, x, y, e, s, name0, name1, name2)
char x[], y[], *s;
int e, namex[], namey[], name0[], name1[], name2[];
{makenul(name2);
zfldset(x, namex, name2);
if (s)
```

```
        {if (eql(s, "Ssim{"))
                {exlists(name1, name2);
                union2(name1, name2);
                makenul(name1);
                zfldset(s, name2, name1);
                union2(name1, name2);}
        else
        if (eql(s, "Simp{"))
                {exlists(name1, name2);
                chngz(name1, name2, s);
                exlists(name1, name2);
                union2(name1, name2);}}
else
if (!e) {chngz(name2, name1, y);
        exlists(name1, name2);}
makenul(name1);
prodset(name2, namey, name1);
if (cntset(name1))
        {putop(namex, namey, name0, "OP{");
        if (e) putop(namey, namex, name0, "OP{");}
}/*end of rxy2n(namex, namey, x, y, e, s, name0, name1, name2)*/

rxyn1(p11, page, line, x, y, e, s, name0, name1, name2)  char x[], y[], *s;
int e, p11[], page, line, name0[], name1[], name2[];
{int j, p12[2], x2[4], y2[4];
if (page < 0) return;  /*list is empty*/
setpage(page);  getlink(line, x2, y2);
p12[0] = x2[0];  p12[1] = y2[0];
if (!e  && (p11[0] != p12[0] || p11[1] != p12[1])
|| (e && (p11[0] > p12[0] || p11[0] == p12[0] && p11[1] > p12[1])))
rxy2n(p11, p12, x, y, e, s, name0, name1, name2);
j = 4;  while(--j)  if (x2[j] >= 0)
rxyn1(p11, x2[j], y2[j], x, y, e, s, name0, name1, name2);
}/*end of rxyn1(p11, page, line, x, y, e, s, name0, name1, name2)*/

rxy21(page1, line1, page2, line2, x, y, e, s, name0, name1, name2)
int e, page1, line1, page2, line2, name0[], name1[], name2[];
char x[], y[], *s;
{int j, p11[2], x1[4], y1[4];
if (page1 < 0) return;  /*first list is empty*/
setpage(page1);  getlink(line1, x1, y1);
p11[0] = x1[0];  p11[1] = y1[0];
rxyn1(p11, page2, line2, x, y, e, s, name0, name1, name2);
j = 4;  while (--j) if(x1[j] >= 0)
rxy21(x1[j], y1[j], page2, line2, x, y, e, s, name0, name1, name2);
}/*end of rxy21(page1, line1, page2, line2, x, y, e, s, name0, name1, name2)*/

prxy(x, y, e, s)  char x[], y[], *s;  int e;
{int xp1[2], x1[4], y1[4], x2[4], y2[4], name0[2], name1[2], name2[2];
if (!findname(xp1)) return;
makenul(tname0);
setpage(xp1[0]);  getlink(xp1[1], x2, y2);
/*at some future time these may be different*/
setpage(xp1[0]);  getlink(xp1[1], x1, y1);
if (x1[1] >= 0 && x2[1] >= 0)
rxy21(x1[1], y1[1], x2[1], y2[1], x, y, e, s, tname0, tname1, tname2);
```

```
exlists(ppnt, tname0);
printf("count: %d\n", cntset(ppnt));
makenul(tname1);
makenul(tname2);
}/*end of prxy(x, y, c, s)*/

pexchng()
{int xpl[2];
if (!findname(xpl)) return;
exlists(xpl, ppnt);
}/*end of pexchng()*/

pelim()
{int flg, j;
getword(str, 64, &chlast);
if (eql(str, "field")) flg = 1;  else
if (eql(str, "fieldnot")) flg = 0;  else
        {printf(nocapis);  get012;  return;}
j = getstr(str, 64, &chlast);
if (str[j-1] != '{' && !eql(str, "TXT:"))
        {printf(str);
        printf("{ do you mean? . .  repeat request:\n");
        get012;  return;}
makenul(tname0);
zfldset(str, ppnt, tname0);
if (!flg) {exlists(ppnt, tname0);  return;}
exlists(ppnt, tname1);  makenul(ppnt);
difset(tname1, tname0, ppnt);
makenul(tname1);
}/*end of pelim()*/

swtch()
{switch(str[0]) {
case 'A': if (eql(str,"ABconnect"))     pxycon0("A{","B{");      else
          if (eql(str,"ABiterate"))     pxyiter("A{","B{");      else
          if (eql(str,"ABpath"))        pxypath("A{","B{");      else
case 'a': if (eql(str,"add"))   padd(); else
          if (eql(str,"attach"))        pattach();       else
case 'B': if (eql(str,"BAconnect"))     pxycon0("B{","A{");      else
          if (eql(str,"BAiterate"))     pxyiter("B{","A{");      else
          if (eql(str,"BApath"))        pxypath("B{","A{");      else
          if (eql(str,"BBconnect"))     pxycon0("B{","B{");      else
          if (eql(str,"BBpath"))        pxypath("B{","B{");      else
          if (eql(str,"BBiterate"))     pxyiter("B{","B{");      else
case 'c': if (eql(str,"count")) pcount();       else
          if (eql(str,"create"))        pcreate();       else
case 'd': if (eql(str,"delete"))        pdelete();       else
          if (eql(str,"display"))       pdisplay();      else
case 'e': if (eql(str,"exchange"))      pexchng();       else
          if (eql(str,"eliminate"))     pelim();         else
case 'h': if (eql(str,"help"))  while(describe("inter.alf"));   else
case 'i': if (eql(str,"intersect"))     pinter();        else
          if (eql(str,"is"))    pis();  else
case 'm': if (eql(str,"make"))  pmake();        else
case 'o': if (eql(str,"on"))    pon();  else
case 'p': if (eql(str,"point")) ppoint();       else
```

```
case 'R': if (eql(str,"Rexp"))  prxy("A{","B{", 0, 0);  else
          if (eql(str,"Rcon"))  prxy("L{","B{", 1, 0);  else
          if (eql(str,"Rhom"))  prxy("A{","A{", 1, 0);  else
          if (eql(str,"Rgen"))  prxy("A{","A{", 0, "Simp{");    else
          if (eql(str,"Rexa"))  prxy("B{","A{", 0, "Simp{");    else
          if (eql(str,"Rmas"))  prxy("A{","A{", 1, "Ssim{");    else
case 'r': if (eql(str,"relation"))       prelation();    else
          if (eql(str,"remove"))         premove();      else
case 'u': if (eql(str,"union")) punion();       else
default:        {printf(nocapis);  get012;}}
}/*end of swtch()*/

main()
{int j;
intinit();
while(!eql(str, "halt"))
        {if (flgp && !eql(str, "create") && !eql(str, "point"))
                {printf("  P or pointer register not set yet\n");
                get012;}
        else swtch();
        if (oldtim1)
                {times(timex);  printab(7);
                printf("%d      %d\n",
                timex[0] - oldtim0, timex[1] - oldtim1);
                oldtim0 = timex[0];  oldtim1 = timex[1];}
        printf(">");  flushbuf();  getword(str, 64, &chlast);}
flushbuf();
}/*end of main()*/
```

```
#define lmax 128
#define getcr while (getchar() != 012)
#define wch nchar[(j < 512) ? (j++) : wrt(&j, nchar)]



/*      mp.c

/*      mp.c contains a collection of procedures and a main
/*      for processing the modules (stite.mod etc..  )

/*      needs the following files:  cf.o, u.o, hl.o, lp.o

*/


char *checkx;
char ch[lmax];
int m0, debug04, debug05, flmods, mlast;
int pcmod[2], modname[2];
extern hash01();          extern putitem();         newlink();
extern rdchfl();          extern writech();         extern readint();
extern error();           extern flushbuf();        extern setlink();
extern lnkinit();         extern chinit();
extern getjch();          extern getpgch();         extern gtpgfnu();
extern char chbufxxx[512];
extern errno;
extern int debug01, debug02, debug03;

chread()
{int c;  static int j;
c = rdchfl(pcmod, chbufxxx, flmods);
/*for debug do*/
if (debug05)
if (j > 72 && c <= ' ' || c == 012)
        {putchar(012);  j = 0;}
else
        {putchar(c);   j =+ 1;}
/*end debug*/
return(c);
}/*end of chread()*/

initmod()
{flmods = open("stite.mod", 0);
if (errno) printf("\n errno = %d ", errno);
 m0 = 0;
pcmod[0] = (-1);  pcmod[1] = 512;
checkx = &"ABCFHD";
printf("\n number of last module: ");
mlast = readint(0);
return(012);
}/*end of initmod()*/

dump(chs, j)  int j;  char chs[];
{int k;
```

```
k = (-1);  while (k <= j) putchar(chs[++k]);
}/*end of dump(chs, j)*/

replace(a, b, str)  int a, b;  char *str;
{int c;
while (c = *str++) if (c == a) *(str-1) = b;
}/*end of replace(a, b, str)*/

int wrt(j, nchar) int *j;  char nchar[];
{static int p, flsrt;
if (!p) flsrt = creat("srt.d", 0600);
seek(flsrt, p++, 3);
write(flsrt, nchar, *j);
*j = 1;
return(0);
}/*end of wrt(j, nchar)*/

wsort(fld, mod)  int fld[], mod[];
{int temp;  static int j;  static char nchar[512];
if (fld[1] >= 42) {wrt(&j, nchar);  return;}
wch = fld[1] + 041;
wch = ((temp = fld[0]) & 037) + 041;
wch = ((temp =>> 5) & 037) + 041;
wch = ((temp >> 5) & 037) + 041;
wch = mod[1] + 041;
wch = ((temp = mod[0]) & 037) + 041;
wch = ((temp =>> 5) & 037) + 041;
wch = ((temp >> 5) & 037) + 041;
wch = 012;
}/*end of wsort(fld, mod)*/


putfld()
{int fldname[2];
hash01(ch, fldname);
putitem(fldname[0], fldname[1], modname);
/* if no sort:
putitem(modname[0], modname[1], fldname);
*/
/* to sort:
*/
wsort(fldname, modname);
}/*end of putfld()*/

getmodn(chl)  int chl;
{int m, l, k;
l = (-1);  while (l < 3) ch[++l] = chread();
while
(chl != 012
|| ch[0] != 'M'
|| ch[1] != 'O'
|| ch[2] != 'D'
|| ch[3] != ':')
        {chl = ch[0];
        l = 0;  while (l < 3) {ch[l] = ch[l+1];  l =+ 1;}
        ch[l] = chread();}
```

```
ch[l] = '{';  /*l == 3 now*/
chl = ' ';  while (chl <= ' ') chl = chread();
m = 0;  ch[++l] = chl;
while (060 <= chl  && chl <= 071)
        {m = m * 10 + (chl - 060);
        ch[++l] = chl = chread();}
/*ch[4] thru ch[l-1] contain number*/
if (m != m0 + 1)
        {printf("\n module number not in order: last = %d next = %d ", m0, m);
        k = 0;  while(k <= l) putchar(ch[k++]);}
m0 = m;
ch[l+2] = ch[l];
ch[l] = '}';
ch[++l] = 0;
/*for debug do*/
if (debug04)
        {printf("\n end of getmodn %d ", m);
        dump(ch, l);  getcr;}
/*end debug*/
hash01(ch, modname);
/*m0 contains module number,
ch[0] thru ch[l] contains null terminated string,
ch[l+1] contains next charachter*/
return(ch[l+1]);
}/*end of getmodn(chl)*/

gettext(chl)  int chl;
{int l, pl[2], pagech0, jch0;
l = (-1);  while (l < 3) ch[++l] = chread();
while
(chl != 012
|| ch[0] != 'T'
|| ch[1] != 'X'
|| ch[2] != 'T'
|| ch[3] != ':')
        {chl = ch[0];
        l = 0;  while (l < 3)  {ch[l] = ch[l+1];  l =+ 1;}
        ch[l] = chread();}
jch0 =  getjch();
pagech0 = getpgch();  /* record jch & pagech*/
writech(ch[0]);
l = 0;  while (l < 3) {ch[l] = ch[l+1];  l =+ 1;}
while
(ch[0] != 012
|| ch[1] != 'A'
|| ch[2] != ':')
        {writech(ch[0]);
        l  = 0;  while (l < 2) {ch[l] = ch[l+1];  l =+ 1;}
        ch[l] = chread();}
writech(0);
/*fix up links for text*/
newlink(pl);
setlink(pl[1], pagech0, jch0>>1, -1, -1, -1, -1, -1, -1);
putitem(pl[0], pl[1], modname);
return(ch[l+1]);
}/*end of gettext(chl)*/
```

```
getxfld(x)  int x[];
{int l, chl;
ch[0] = x[0];
ch[1] = '{';
chl = ' ';  while (chl <= ' ') chl = chread();
l = 2;  ch[2]= chl;
while
(ch[l] != ',' &&
(ch[l-2] != 012
|| ch[l] != ':' ))
        {if (l < lmax - 4) l =+ 1;
        ch[l] = chread();}
/*for debug do*/
if (debug04)
        {printf("\n this is getxfld %c %d %c %c ",
        x[0], l, ch[l-1], ch[l]);  getcr;}
/*end debug*/
if (ch[l] != ',')
        {x[0] = ch[l-1];  l =- 2;}
while (ch[--l] <= ' ');
ch[++l] = '}';
ch[++l] = 0;
replace(012, 040, ch);  /*remove carriage returns*/
/*null terminated string is now in ch[0] thru ch[l]*/
/*ok fields A, B, C, F & H but not D */
/*for debug do*/
if (debug04)
        {printf("\n end getxfld ");
        dump(ch, l);  getcr;}
/*end debug*/
putfld();
return(ch[l+1]);
}/*end of getxfld(x)*/

getlast(x) int x[];
{int chl, l;
ch[0] = x[0];
ch[1] = '{';
chl = ' ';  while (chl <= ' ') chl = chread();
l = 2;  ch[l] = chl;
while(ch[l] != 012)
        {if (l < lmax - 4) l =+ 1;
        ch[l] = chread();}
while (ch[--l] <= ' ');
ch[++l] = '}';
ch[++l] = 0;
ch[l+1] = 012;
/*for debug do*/
if (debug04)
        {printf("\n end of getlast %c  ", x[0]);
        dump(ch, l);  getcr;}
/*end debug*/
/*null terminated string is now in ch[0] thru
ch[l], ch[l+1] == 012 */
putfld();
```

```
return(ch[l+1]);
}/*end of getlast(x)*/

main()
{int j, c, x[1];
/*for debug do*/
printf("\n debug01 = ");  debug01 = readint(0);
printf("\n debug02 = ");  debug02 = readint(0);
printf("\n debug03 = ");  debug03 = readint(0);
printf("\n debug04 = ");  debug04 = readint(0);
printf("\n debug05 = ");  debug05 = readint(0);
/*end debug*/
chinit();
lnkinit();
c = initmod();
while (m0 < mlast)
        {c = getmodn(c);
        c = gettext(c);
        j = 0;  x[0] = 'A';
        while (x[0] != 'D')
                {c = getxfld(x);
                if (x[0] != checkx[j])
                        {j =+ 1;
                        while (x[0] != checkx[j] && j)
                                {printf(

"\nfield %c missing or out of order, field %c is next;  mod = %d ",

                                checkx[j], x[0], m0);
                                if (++j > 5) j = 0;}}}
        c = getlast(x);
        if (debug05 || debug04)
                {flushbuf();
                printf("\n continue ? ");  getcr;}}
flushbuf();
printf("\n last mod = %d ", m0);
printf("\n pagech = %d  pgfnu = %d ", getpgch(), gtpgfnu());
pcmod[l] = 42;  wsort(pcmod, modname);
}/*end of main()*/
```