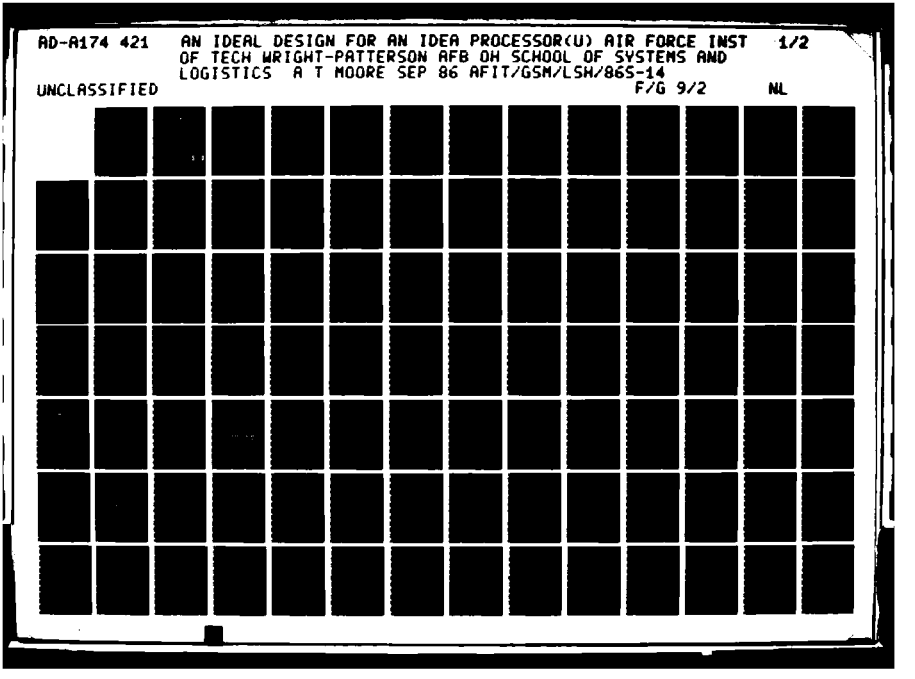
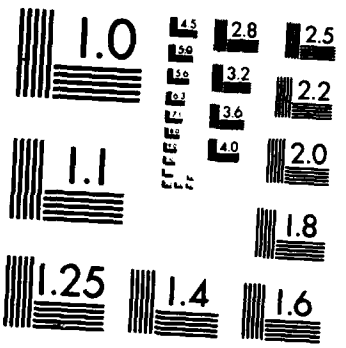
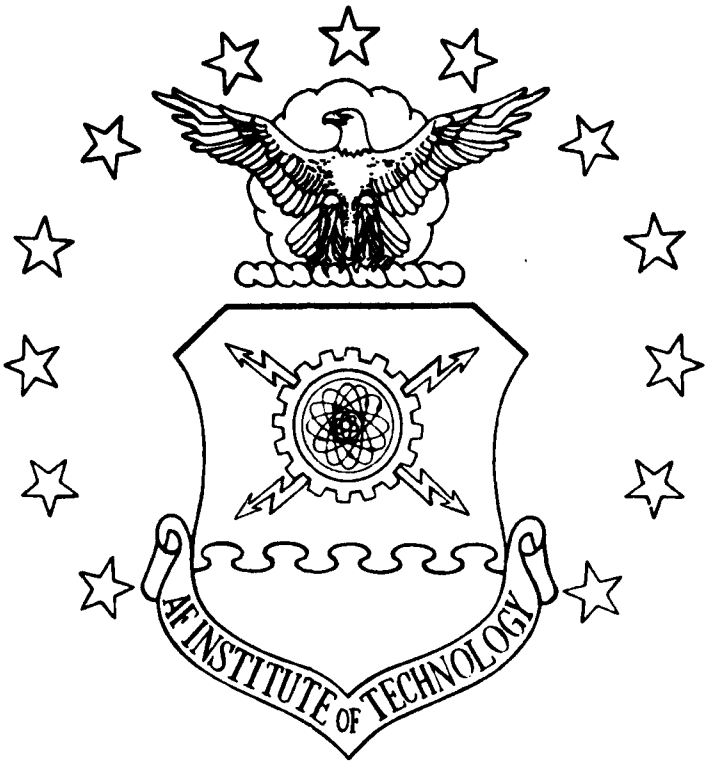

AD-A174 421 AN IDEAL DESIGN FOR AN IDEA PROCESSOR(U) AIR FORCE INST 1/2
OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF SYSTEMS AND
LOGISTICS A T MOORE SEP 86 AFIT/GSM/LSH/86S-14
UNCLASSIFIED F/G 9/2 NL





AN IDEAL DESIGN FOR

AN IDEA PROCESSOR

THESIS

Andrew T. Moore
Captain, USAF

AFIT/GSM/LSH/86S-14

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

2

AFIT/GSM/LSH/86

AN IDEAL DESIGN FOR

AN IDEA PROCESSOR

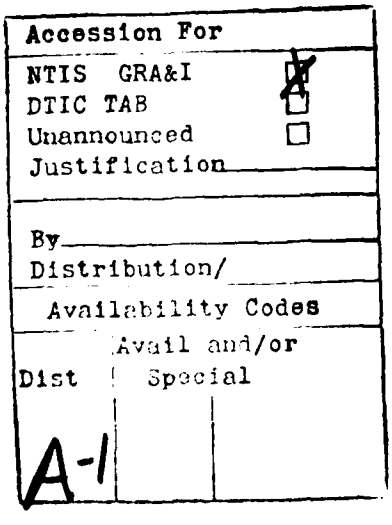
THESIS

Andrew T. Moore
Captain, USAF

AFIT/GSM/LSH/86S-14

Approved for public release; distribution unlimited

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information is contained therein. Furthermore, the views expressed in the document are those of the author and do not necessarily reflect the views of the School of Systems and Logistics, the Air University, the United States Air Force, or the Department of Defense.



AFIT/GSM/LSH/86S-14

AN IDEAL DESIGN FOR AN IDEA PROCESSOR

THESIS

Presented to the Faculty of the School of Systems and Logistics

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Systems Management

Andrew T. Moore, B.S.

Captain, USAF

September 1986

Acknowledgements

The background for this effort comes from two very dissimilar sources. Douglas Hofstadter, in his book Godel, Esher, Bach: An Eternal Golden Braid, and in The Mind's Eye (with Daniel Dennett), ignited my interest in conceptual aspects of the human mind. On the practical side, it was while working on the software specifications and Statement of Work for the Infrared Search and Track (IRST) program that I first saw the need for a hierarchical text editor of some kind. For this "real world" experience I thank the people of ASD/RW.

A thesis on such a unique subject as idea processors could have been accomplished only with the advice and encouragement of many people. I sincerely appreciate the help I have received, either directly or indirectly, from the instructors and staff of the School of Systems and Logistics, as well as my friends and family.

I wish to thank Dr. Terrance Skelton for the significant contributions he made to this thesis. I also received valuable advice and opinions from Captain Thomas Triscari and Mr. Gary Morrison at various points during my research. Additionally, Mr. Jim Meadows helped me with critical documentation on the Burroughs hardware and software.

The eight Air Force officers who participated in this research gave me more than just a few hours on a weekend; they gave me their thoughts and expert opinions as professional communicators. I am indebted to each of them for their honesty, open-mindedness, and positive attitude.

Throughout this thesis I asked my wife Deanna for her patience and understanding. She gave me both and much more, and I came to rely on the common sense of her comments and opinions. To her and to my son Brian I dedicate this thesis.

Finally, I wish to thank Dr. Charles Fenno for his expert guidance and stimulating thoughts. It was he who planted the seed idea for this thesis, he who watched patiently as it grew, and he who pruned the branches to make it a strong, healthy tree--all this, even as he was busily tending other plants in the garden. I will always remember his unique blend of experience, precision, humor, positive attitude, and dedication.

Table of Contents

List of Figures

List of Tables

Abstract

This ~~research~~ determined requirements for an idea processor, a computer program that supports the process of composition by helping the user generate, organize, translate, and review ideas as they are prepared for written or verbal communication. These requirements were derived from theories and models of composition found in a review of current literature. An initial design for an idea processor was developed to meet these requirements, based on features of existing software packages and on the hardware capabilities of microcomputers in use in the Air Force, such as the Zenith Z-100. To implement this initial design, a prototype idea processor was developed on the Burroughs B-26 microcomputer system at AFIT.

The original requirements and design were validated in an exploratory experiment. Eight graduate students--identified by communication instructors as "expert" communicators--performed example idea processing tasks using the prototype idea processor. These participants verbalized any perceived "obstacles" to their own preferred style of composition. The participants' comments were recorded and classified according to four types of obstacles: missing

requirements (type 1), incomplete design (type 2), faulty implementation (type 3), or the limitations of each person's writing ability or the testing environment (type 0).

The original idea processor design was modified to remove the type 2 obstacles found in the validation. Solutions to other obstacles and general comments by the participants were presented as suggested improvements to the prototype idea processor, and directions for further research were discussed.

AN IDEAL DESIGN FOR AN IDEA PROCESSOR

I. Introduction

In recent years computers have been used to help prepare various forms of communication. Word processors have made editing and formatting of final documents much easier and simpler. In other areas of composition, however, computer support is much less well developed--generating and organizing ideas is still largely a manual human activity. Recent attempts to provide computer support to these kinds of communication activity have led to a new concept in software--idea processors.

An idea processor is just one example of how computers can be used earlier in the composition process. An idea processor provides a workspace for ideas: a place to put down thoughts during brainstorming; a place to organize those thoughts as they become more refined; a place to add text as thoughts are translated into language; and a place to make final changes in the overall structure of the thoughts. An idea processor would augment, rather than replace, a word processor.

What should an idea processor do? What functions must it perform? How should it organize information? Because idea processors are so new, there are not yet any clear

answers to these questions. This thesis represents a first step in defining the bounds from within which an idea processor must work in order to be effectively used.

Research Objectives

The purpose of this research is to determine the requirements for an idea processor to be used by communicators, and from these requirements to design an idea processor that could be implemented on standard Air Force microcomputers. These two major objectives are achieved through the following subobjectives:

(1) identify relevant theories of composition from current literature,

(2) define a good communicator's requirements for an ideal idea processor,

(3) establish a design for an idea processor that can be implemented on Air Force microcomputers, and

(4) validate and revise this design based on exploratory research.

Definitions

Composition is the process of preparing to communicate some specific information by collecting and organizing the ideas, concepts, and data in that information and transforming them into written or spoken language. As later sections of this report substantiate, cognitive science research has found that composition involves the following activities as a minimum: generating ideas (brainstorming and researching), organizing ideas (manipulating, grouping,

and ordering), translating ideas (into acceptable language), and reviewing ideas (evaluating and editing).

An idea processor is a computer program that supports the process of composition by helping the user generate, organize, translate, and review ideas.

Requirements are defined in this report as the minimum capabilities that must be included for a system to effectively perform its intended function.

A design is a set of specific statements describing the structure and functions of a system.

An obstacle is used in this report to describe any perceived problem or difficulty that prevents a communicator from accomplishing the task he or she wants to do in a comfortable and natural manner.

Investigative Questions

Composition. What is currently known about how people compose, the structures they use to represent ideas, and the mental processes that are involved in composition?

Requirements. What does an idea processor need to do to support composition? In what formats should the idea processor store information entered by the user? What operations does the idea processor need to be able to perform on ideas in order to support composition? Upon what underlying model(s) of composition should the idea processor be based? What composition styles should the idea processor accommodate?

Design. How should the user's ideas be displayed? What specific functions (add, delete, change, move, copy, etc.) should be performed by the idea processor? How should these functions be initiated by the user (e.g. numeric or cursor menus, function keys, control key sequences, command words)?

Scope and Limitations

Approach. There are three possible points of view to take in studying the process of composition: how people should compose (prescriptive), how people think they compose (intuitive), and how people actually compose (descriptive). In this research the primary approach has been descriptive, with some attention given to the intuitive approach. Although the prescriptive approach could have been taken by incorporating rhetorical heuristics (such as those offered by Aristotle) into the design of an idea processor, this research instead assumed that the user was the "expert" and would compose using whatever skills he or she already knew. This research sought to identify a design for an ideal idea processor that would help the user compose written and oral communication in whatever way each user sees fit.

Research Goals. The primary purpose of this research was to establish requirements and design for an idea processor to be used in the Air Force. No attempt was made to evaluate existing software for suitability or

performance, to collect experimental data on the cognitive processes involved in composition, or to conduct operational tests of Air Force personnel using idea processors in their daily work. These activities were beyond the scope of this investigation and were better left for future research, as discussed in Chapter VI.

II. Methodology

Introduction

The first step in meeting the research objective of defining basic requirements for an idea processor was to review current literature. Research in cognitive science and psychology, technical writing and communication, and human factors engineering has yielded a small but powerful set of theories on how the human mind composes. From these theories a set of basic requirements for an idea processor was derived.

Also identified in the literature review was information about currently available idea processor software. General features of these "outline generators" were evaluated for consistency with current composition theories. The features which were validated by the research literature were used in this study as the basis for an ideal idea processor design. This design was then adjusted to take into account the limitations imposed by standard Air Force microcomputers, such as the height and width of the screen, color or graphics capability, amount of memory, and type of input and storage devices.

Based on this ideal design, a prototype idea processor was implemented on the Burroughs B-26 microcomputer. Eight

graduate students participated in an exploratory prototype validation experiment that involved the use of sample idea-processing tasks. All participants performed the tasks using the prototype idea processor, while their comments and actions were recorded. These observations were classified according to different types of obstacles encountered by the participants. The basic requirements and ideal design were modified based on the feedback provided by the validation.

Requirements

Literature Review. No one single field of study covers all aspects of the composition process. Psychology and human factors studies traditionally cover more specific processes of the human mind, such as memory and perception. Research on communication is typically more broad and applied. The relatively new field of cognitive science, itself a multidisciplinary science, yielded the most relevant material on how people prepare communication. Literature on all of these fields was reviewed for current information.

Although a great deal of information related to composition was found, only those theories directly relevant to composition were considered in this investigation. These theories, and the requirements they imply for an idea processor, are discussed in detail in the next chapter. The

requirements resulting from the literature review identify what an ideal idea processor must do in order to be useful to someone preparing communication.

Design

Existing Software. In addition to research on current theories of composition, the literature contains reviews of several software packages that could fulfill some of the requirements of an idea processor. The general designs and features of these "outline generators" were evaluated in light of current theories of composition. Those features that were supported by the composition theories and that appeared in several software packages were combined into an initial design for an idea processor.

Initial Design. While the requirements specify "what" an idea processor must do, the design specifies "how" the requirements will be met. The initial design for an ideal idea processor was derived from features of existing software packages, but eventually evolved into a new, independent design. This design emphasized a small set of simple but powerful functions that worked in a consistent fashion.

Hardware Considerations. Since one objective of this research was to determine the design for an idea processor that could be used in the Air Force, the initial design had

to be evaluated in terms of the standard microcomputers currently available in the Air Force.

This research was conducted with the assumption that the microcomputer system on which an idea processor would be implemented would have the following minimum characteristics:

- A standard monochrome alphanumeric display, 80 columns by 25 lines, with the capability to display any character in either normal or some alternate mode (reversed, half-intensity, bold, italicized, underlined). No graphics or color capabilities were assumed.
- A standard alphanumeric keyboard with some method for selecting commands (function keys, alternate keypad mode, control or escape characters) as the only input device. No positional control, such as joystick, light pen, mouse, or touchscreen, was assumed.
- A mass memory storage device (floppy or hard disk drive, tape drive), and access to a printer.

These characteristics are likely to be found on virtually all microcomputer systems currently in use by the Air Force, including the Zenith family (Z-100, Z-120, Z-150), the IBM PC family, and the Burroughs B-20 family (B-26, B-29). As a result of the limitations imposed by these systems, a more specific design resulted from the original "ideal" design. The details of this design are discussed in Chapter IV.

Implementation

The design derived in this thesis was used to develop more detailed specifications for an idea processor to be

implemented on the Burroughs B-26 microcomputer system at AFIT. A first prototype was developed in the BASIC programming language to explore implementation possibilities and difficulties. The BASIC language was chosen because it was interpreted (executed directly as source code), without the preprocessing necessary for a compiled language. This allowed initial programs to be developed and modified rapidly and flexibly. A second prototype was developed in the programming language Pascal over a period of several months. This prototype implemented most features of the design developed as part of this thesis. Pascal was chosen as the final development language for the following reasons:

(1) Pascal's highly modular, structured form supports development of more standardized, portable code,

(2) as a compiled language, it would eventually run faster, and

(3) the researcher had a great deal of experience with the language.

While the Pascal code cannot simply be transferred to another Air Force standard microcomputer (such as the Zenith Z-100), the code that would have to be changed is minimal. Hardware- and compiler-dependent portions of the prototype were kept intentionally separate and easily identified. The Pascal source code for the prototype idea processor is listed in Appendix A.

Validation

The idea processor design developed as part of this research was hypothesized based on current theories of composition and existing software packages. Some means of testing this design, and the requirements from which it was derived, was needed. A strict, scientific experiment was not feasible for several reasons: (1) the researcher could not identify measurable variables that were relevant to the quality, performance, and suitability of an idea processor design; (2) no previous research--conducted specifically on idea processors--existed to serve as a precedent for this research; and (3) there was insufficient time to design and execute a comprehensive experiment on the cognitive processes involved in composition, and then apply the results to the evaluation of this idea processor design.

Given these constraints, the best approach to validating the idea processor design was to perform exploratory research by having several people actually use an idea processor based on the design in question. Rather than measure specific variables, this exploratory validation would make more general observations, depending largely on observation and on verbal comments by the participants.

Task Development. An optimum communication task for evaluating the design of an idea processor would have

involved asking each test subject to compose original documents. However, it was not practical to ask volunteers to compose lengthy documents in a testing environment, or to recall and record either actual or imaginary data (as in the generating phase of composition). The approach taken in this research was to modify existing documents to represent unstructured research notes. This approach allowed participants to bypass the generation stage of composition, and standardized the basic ideas they worked with during the validation. No effective way was found to test the generating phase of composition within the scope of this research.

The idea processing tasks used in this research involved relatively small amounts of information on general topics, as taken from encyclopedia articles. The topics were chosen to minimize discipline-specific bias based on specific knowledge that each participant would bring to the validation. These well-structured documents were scrambled to produce sets of seemingly random data. Thus, the participants were asked to organize (move, group, and order) and review (evaluate and edit) the information presented in the tasks. The tasks are listed in Appendix B.

Participant Selection. Eight volunteer graduate students from the AFIT School of Systems and Logistics were selected as participants in a validation exercise involving

the prototype idea processor. These participants, identified as proficient by their communication instructors, were treated as experts for validation purposes. Any operation they wanted to do while performing the tasks was considered a necessary part of composition, by definition.

Validation. A validation session was conducted for each of the eight people selected to participate. All validations were conducted at the Air Force Institute of Technology (AFIT) School of Systems and Logistics, in the offices of the Department of Research and Communication (LSH). These sessions began with a brief orientation and training on the prototype, followed by as much practice time as needed for the participant to feel comfortable with the idea processor. Then the participant used the prototype idea processor to complete two representative communication tasks.

The participants were observed and their comments recorded, with particular emphasis on any perceived obstacles to a natural composition process. The presence of the researcher during the validation, while it did represent a form of intrusion on the environment of the participant, was considered acceptable for exploratory research. The participants were asked to discuss their overall impressions of the prototype idea processor, including any specific

strengths and weaknesses. The results of this validation are discussed in more detail in Chapter V.

Analysis

The results of the validation were analyzed and classified according to the source of the obstacles recorded: requirements (type 1), design (type 2), implementation (type 3), and testing environment (type 0). Type 1 obstacles were used to call into question the requirements defined from current theories on composition. Type 2 obstacles were used to revise the initial design. Although the implementation of the prototype idea processor was not revised, recommendations were made based on the Type 3 obstacles found. Type 0 obstacles were used to assess the validity of the research findings. Analysis of the validation results is explained in Chapter V.

III. Literature Review

Introduction

Although there has been little specific research on the concept of electronic idea processing, there are substantial amounts of literature that indirectly address the cognitive processes involved in generating, organizing, translating, and reviewing ideas to form prose. The oldest relevant research field is psychology, the study of the human mind and human behavior. This field has contributed much to the general understanding of perception, memory, motivation, and other aspects of human thought. However, the specific activities involved in composing have not been extensively researched in psychology. Similarly, the human factors engineering and man-machine interface disciplines take an applied approach to the study of human perception and behavior, but again seem to have missed composition, concentrating instead on more tangible problems such as the design of optimum controls and displays.

The composition process is more directly addressed by the fields of communication and technical writing. Unfortunately, the majority of literature in this discipline lacks much theoretical base; it consists primarily of "lessons learned" and "rules of thumb." The best literature on composition comes from a relatively new science: cognitive science. This multidisciplinary field brings

elements of psychology, artificial intelligence, philosophy, communication, mathematics, and logic to the study of how the human mind works. Various aspects of the composition process are frequently addressed in cognitive science articles.

In this review of current literature, all of the disciplines discussed above--psychology, human factors, communications, and cognitive science--were examined for theories relevant to composition. No distinction is made between theories that come from different fields of study.

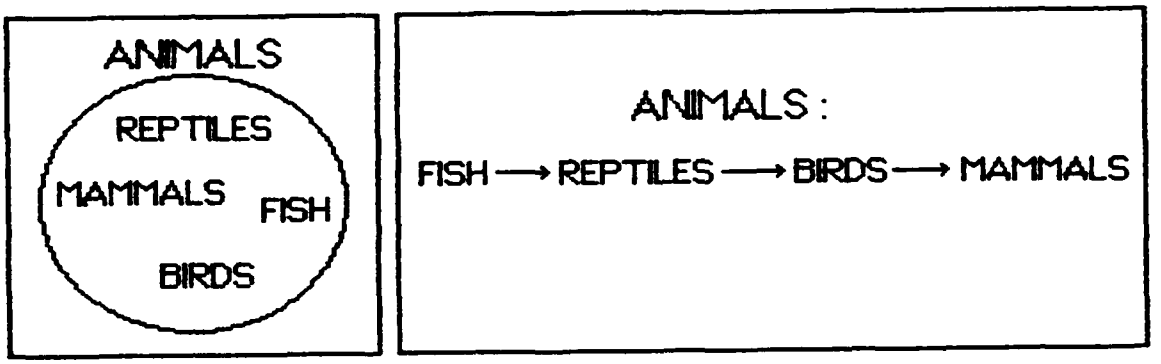
Knowledge Representation

Basic Units. There are many words for the lowest units of knowledge that human beings work with: "data," "objects," or "variables" (22:52). Out of these most fundamental units, the human mind constructs higher level concepts, such as propositions, assertions, and topics. It is these higher level constructs that an idea processor would manipulate. Grunig and others refer to these constructs as "cognitive structures" and define them as

> the abstract patterns in long-term memory that underlie the verbal and visual representations in short-term memory. Cognitive structures contain elements such as subjects or agents, objects, attributes, and recipients of actions, which are connected by spatial, temporal, or logical rules. (13:112)

These constructs are usually represented in language by sentences, paragraphs, and higher divisions (20:349-350).

Lists and Sets. Since most communication consists of more than one idea, it is useful to think in terms of groups of ideas. Two basic types of these groups are sets (13:102) and lists (22:51), as shown in Figures 1 (a) and (b). Sets are "objects or attributes that include or exclude another object or attribute" (13:102); lists are ideas that have a specific order. It is from the building blocks of ideas, lists of ideas, and sets of ideas that more complex cognitive structures are formed.



(a) Set (b) List

Figure 1. Knowledge Structures

Hierarchies. One of the most important structures of knowledge for communication is the hierarchy or tree (16:223). Friendly describes hierarchies as "sets of categories or groups of items arranged in a nested fashion" (8:190). While humans transmit and receive language in a linear or sequential fashion, the concepts they communicate

are often organized hierarchically: an overall topic contains several subtopics, each of which contains more sub-subtopics, and so on (4:64). An example of a hierarchy is shown in Figure 2.

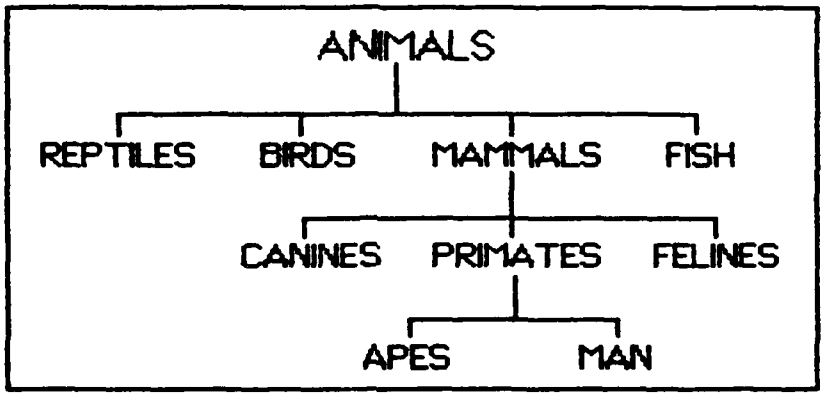


Figure 2. Hierarchy

The hierarchical nature of a document is evident in the form of its table of contents, which is indented to show each section's level in the hierarchy of the document. Harrington and Walton go so far as to advocate using Warnier-Orr diagrams--typically used for documenting the tree-like structure of computer programs--to help structure expository writing (14:197). Figures 3 and 4 present examples of both the table of contents and the Warnier-Orr structures of hierarchical information. It is this underlying hierarchical structure, built up from sets and lists of ideas, that leads to the first requirement for an idea processor:

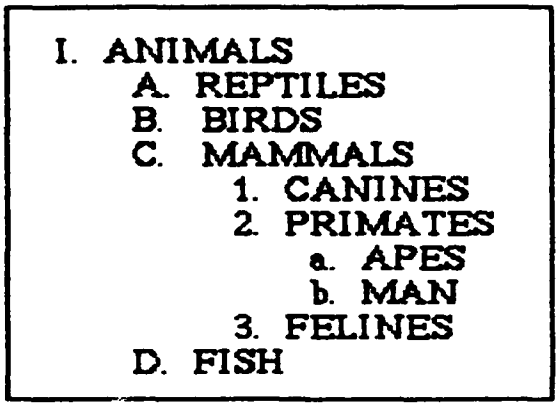


Figure 3. Table of Contents

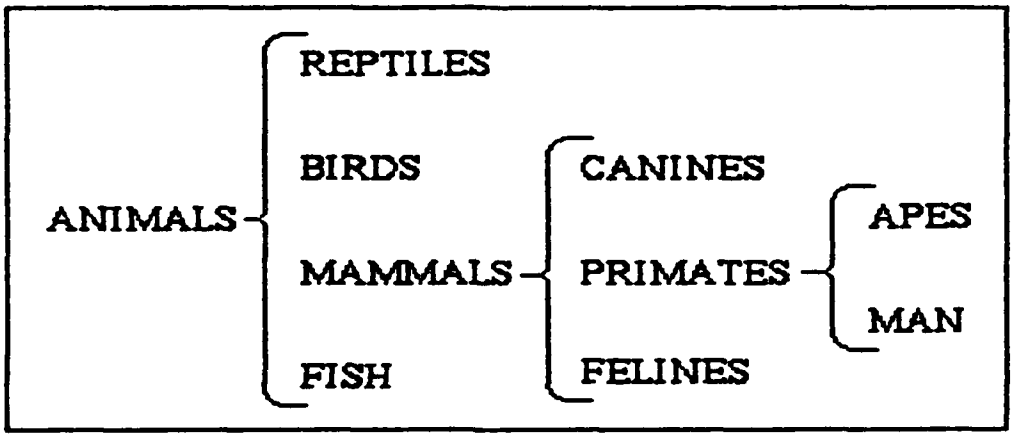


Figure 4. Warnier-Orr Diagram (14:196)

Requirement:	An idea processor must support lists, sets, and hierarchies of ideas.

While the underlying structure of concepts in most forms of communication is hierarchical, the words, sentences, and paragraphs that make up the language of the communication are still sequential. At a fundamental level, a document is a list of sections, each of which is a list of

paragraphs, each of which is a list of sentences, each of which is a list of words. Thus, a document has the dual nature of hierarchical concepts represented as a list of language elements. Peels and others acknowledge the dual nature of a document by making a distinction between its "natural" or "logical" form versus the "physical" form (20:349-350). In a similar way, Harrington and Walton distinguish between thinking about a document "vertically" (logical structure) and "horizontally" (sequential development) (14:197).

An idea processor would support a hierarchical framework of concepts and topics, on which a communicator could "hang" text to make a complete document. The next requirement addresses this dual nature of communication:

Requirement:	An idea processor must allow sequential text to be associated with each idea.

Cognitive Processes in Composition

Hayes and Flower (15:10-30) have proposed a conceptual model of the composition process, shown in Figure 5, that was used as the basis for the requirements and design discussed in this research. This model hypothesizes four basic cognitive activities during composition: generating, organizing, translating, and reviewing. Generating involves capturing--through activities such as brainstorming, memory

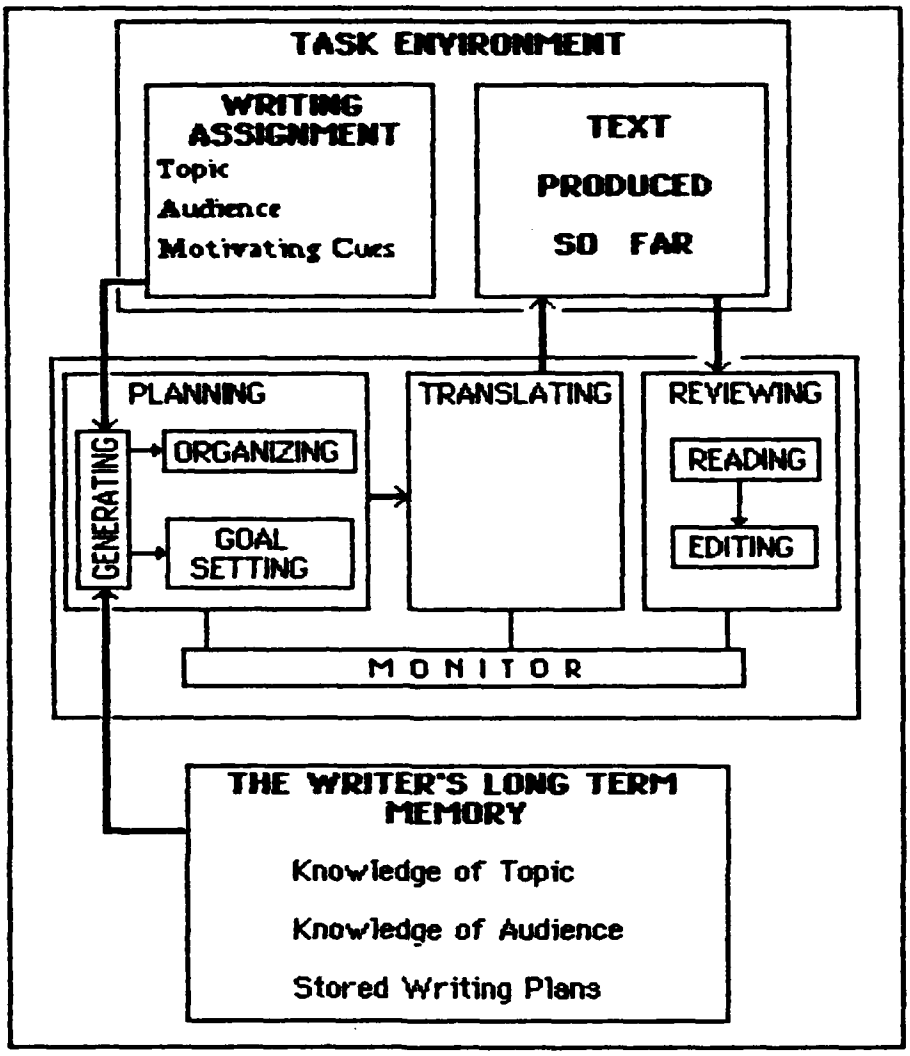


Figure 5. Hayes and Flower's Composition Model (15:11)

recall, and research--ideas and information to be communicated. Organizing encompasses many activities, including manipulation, grouping, and ordering ideas. Translating is the process of transforming concepts and ideas into acceptable spoken or written language. Review involves reading the material that has already been

translated and editing that material by correcting errors and revising (possibly re-translating) selected text.

An important aspect of Hayes and Flower's model is that these four activities can occur in any order, and frequently interrupt each other. A "monitor" function allows a specific activity to temporarily take over, and then returns control to the original activity. An example of this would be a writer thinking of a new idea while trying to write a paragraph; in this case the generating activity interrupts the translating activity.

Collins and Gentner propose writing activities similar to Hayes and Flower's: capturing ideas, manipulating ideas, and producing text. Collins and Gentner group these activities into two major classes: "idea production" and "text production" (4:52). This grouping acknowledges the dual nature of documents previously discussed--that documents consist of ideas and associated text.

Generation. As the first process in composition, generating ideas is perhaps the least well understood, although it is generally agreed that memory recall plays an important part in generating ideas. There are many techniques for prompting the human mind to recall information and generate ideas. The most common of these is "brainstorming," which involves the unrestricted recall and recording of thoughts as they occur, without any evaluation

at all (10:438-439; 12:84). Brainstorming suggests different methods of viewing old ideas to encourage creation of new ideas. An example of a process that may be at work in brainstorming is "coherent deformation," in which "perception and imagination or fact and intuition are combined in language, which then takes on new meaning . . . and leads to new thinking" (19:201).

Regardless of the processes involved in the generation stage of composition, the role of an idea processor is limited to recording ideas as the user brings them to conscious thought:

Requirement: An idea processor must allow easy recording of new ideas.

Organization. Since the ideas and concepts brought forth during generation are rarely in either the hierarchical structure of concepts or the linear structure of a document, some effort must go into organizing these ideas. Hayes and Flower describe the challenge of organization this way:

> When confronting a new or complex issue, writers must often move from a rich array of unorganized, perhaps even contradictory perceptions, memories, and propositions to an integrated notion of just what it is they think about the topic. . . . Much of the work of writing can be the task of transforming incoherent thought and loosely related pockets of information into a highly conceptualized and precisely related knowledge network. (7:34)

Many organizing strategies exist, but manipulation is the means by which these different strategies are carried out. Lower level operations such as insertion, deletion, modification, and movement support higher level organizing methods such as comparison/contrast, dimensionalization, and taxonomization (4:55-58). Before any specific principle of organization can be implemented, the requirement for idea manipulation must satisfied:

Requirement:	An idea processor must allow easy manipulation of ideas, individually or in groups, such that they can be selected, moved, inserted, changed, and removed.

Somewhere between the ages of five and seven years, human beings develop a set of skills that Piaget calls "concrete operations" (21:7-19). Among these operations are two that play an important role in the organization of ideas for communication: class inclusion and serial ordering. Class inclusion is the concept that a set comprising several subsets contains all the objects within those subsets (1:318-319). Once a child develops the concept of class inclusion, he or she understands the hierarchical relationship involved between sets and subsets. Without this operation, a human being would have a very difficult time trying to communicate a complex set of ideas to someone else. He or she would not be able to classify ideas into

topics and subtopics, and address similiar characteristics of each set at the same time.

In the same way that the process of grouping operates on sets of ideas, ordering deals with lists of ideas. A child develops the concrete operation of serial ordering at about the same time that it develops the concept of transitivity--the concept that if A is greater than B, and B is greater than C, then A must be greater than C (1:319). There are many different types of order that can be imposed on a list of ideas: chronological, spatial, cause-effect, general to specific, or least to greatest (for size, weight, degree, importance). By ordering a list of ideas or topics, a communicator makes it easier for the recipient of the communication to follow the communicator's "train of thought" and come to the same conclusions.

Hayes and Flower identify grouping and ordering in their model as elementary operators within the organizing process. They discuss operators such as "order with respect to a previously noted topic," "search for previously noted topics subordinate to present topic," and "identify a category" (15:14). The combined abilities of grouping and ordering are two of the most important idea processing operations that communicators possess.

Requirement:	An idea processor must support grouping (classification) and ordering (prioritization) of ideas.

Translation. The transformation of ideas and concepts into language is a complex process that is only partially understood by cognitive scientists and psychologists. In fact, there is not even agreement on whether thought occurs before language, language occurs before thought, or whether one can exist without the other (13:97-101; 11:122; 3:87-88). Although there are limited aids in the form of spelling/grammar checkers (11:118), the computer is largely unable to help the human in the process of translation, and therefore an idea processor must take a minimal role of text editor during the process of translation.

Requirement:	An idea processor must allow easy insertion and editing of textual information to a specific idea.

Review. It is with the cognitive process of reviewing communication that computers are at their worst and their best. Reviewing involves reading the text that has already been translated. Computer displays are anything but optimum for this task, due to a limited display area and to the direct attention that the user must give issuing commands that cause movement through the document. Fenno calls these difficulties the "Micro View" problem (6:354). On the other hand, the flexibility inherent in computer software allows for the display of both conceptual and textual aspects of documents in a variety of formats. The overall structure of

topics in a document can be provided to the user without the detailed text, a capability that makes the job of insuring coherence and unity much easier (14:197). Computers are also excellent for finding specific words or phrases using a "search" command. And, as with standard word processors, changes to text are easy to make on either an experimental or permanent basis. When this capability for change is applied to the conceptual structure of a document, complete rearrangement of the order of sections and paragraphs can be accomplished easily. The following requirements correspond to the reviewing activities of composition:

Requirement:	An idea processor must allow easy viewing of any and all ideas and their associated text.
Requirement:	An idea processor must allow the user to preview the text associated with ideas in an integrated, sequential manner (as it would appear in a document).

Composition Styles

Hayes and Flower discuss four possible ways in which people might perform the basic composition processes: "Depth first," "Get it down as you think of it, then review," "Perfect first draft," and "Breadth first" (15:20). Each style represents a completely different order and approach to creating communication.

Depth First. The "depth first" writing style describes a communicator who goes through all four composition activities for each sentence until the entire document is complete. The next sentence is not started until the current sentence is a good as it can be. If during composition the organization of the communication is changed, all previous sentences are rewritten to bring the document back into "perfect" shape before the writer goes to the next sentence. Figure 6 is a flowchart that depicts the sequence of actions under the "depth first" writing style.

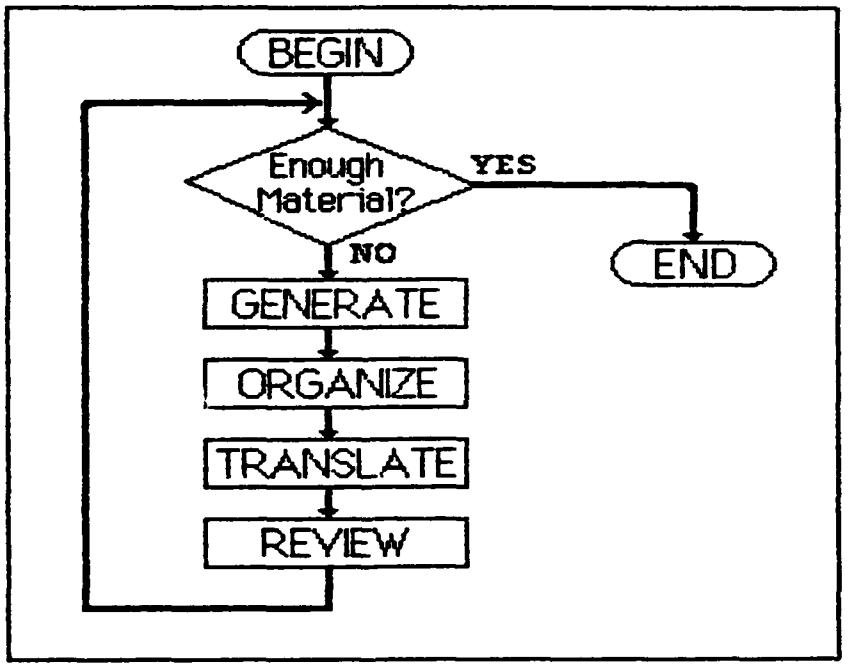


Figure 6. "Depth First" Writing Style

Get It Down. A communicator who uses the "get it down as you think of it, then review" style tries not to worry about the final document while he or she is generating ideas. As shown in Figure 7, each sentence is generated, organized, and translated into language, but the communicator delays evaluation of the material he or she has produced until all ideas have become language. Only then is the document revised and edited.

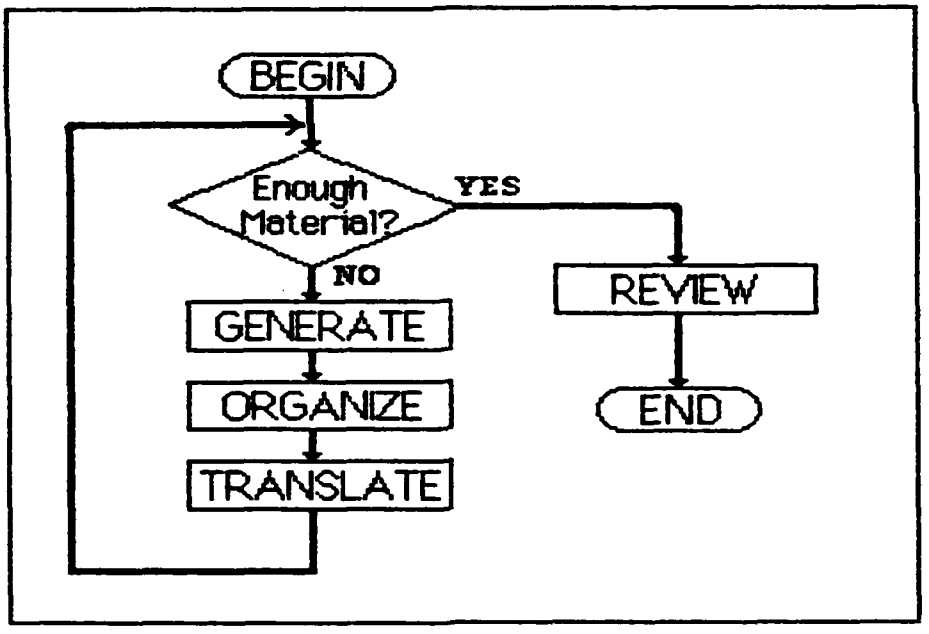


Figure 7. "Get It Down" Writing Style

Perfect First Draft. The "perfect first draft" communicator begins by generating all the ideas he or she can. Then these ideas are organized into an appropriate structure. Finally, the communicator translates and reviews the organized information sentence by sentence so that the first draft is very close to the final document. The flowchart represented by Figure 8 summarizes this style of composition.

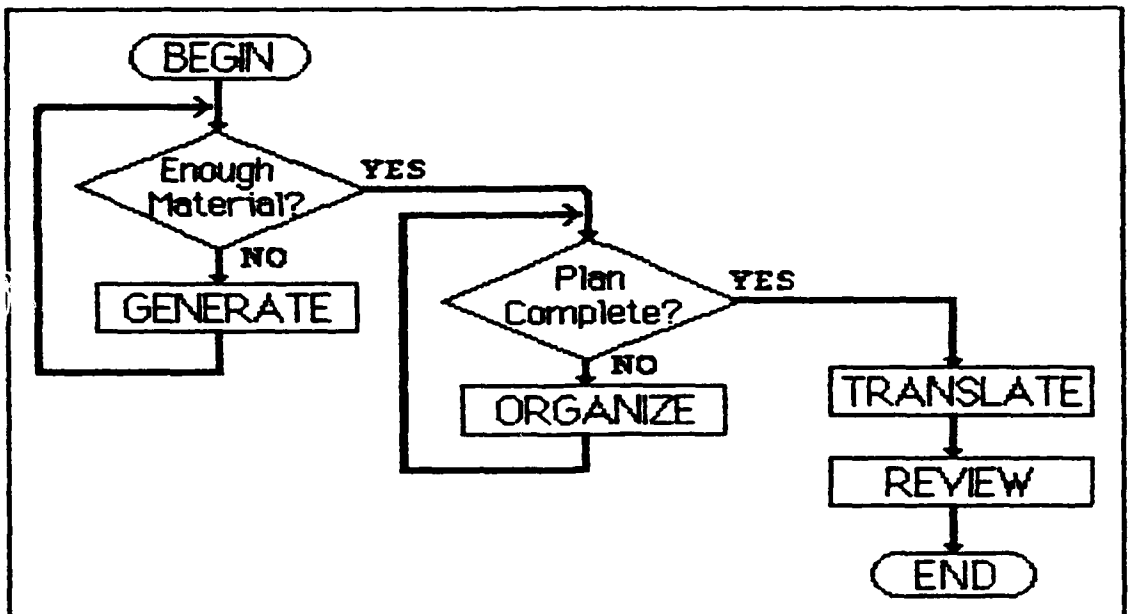


Figure 8. "Perfect First Draft" Writing Style

Breadth First. When a communicator uses the "breadth first" style, he or she always completes each stage of composition before moving on to the next, as shown in Figure 9. Organization is started only after all ideas have been generated; ideas are translated only after they have been completely organized. The document is reviewed as a complete document after all ideas have been translated.

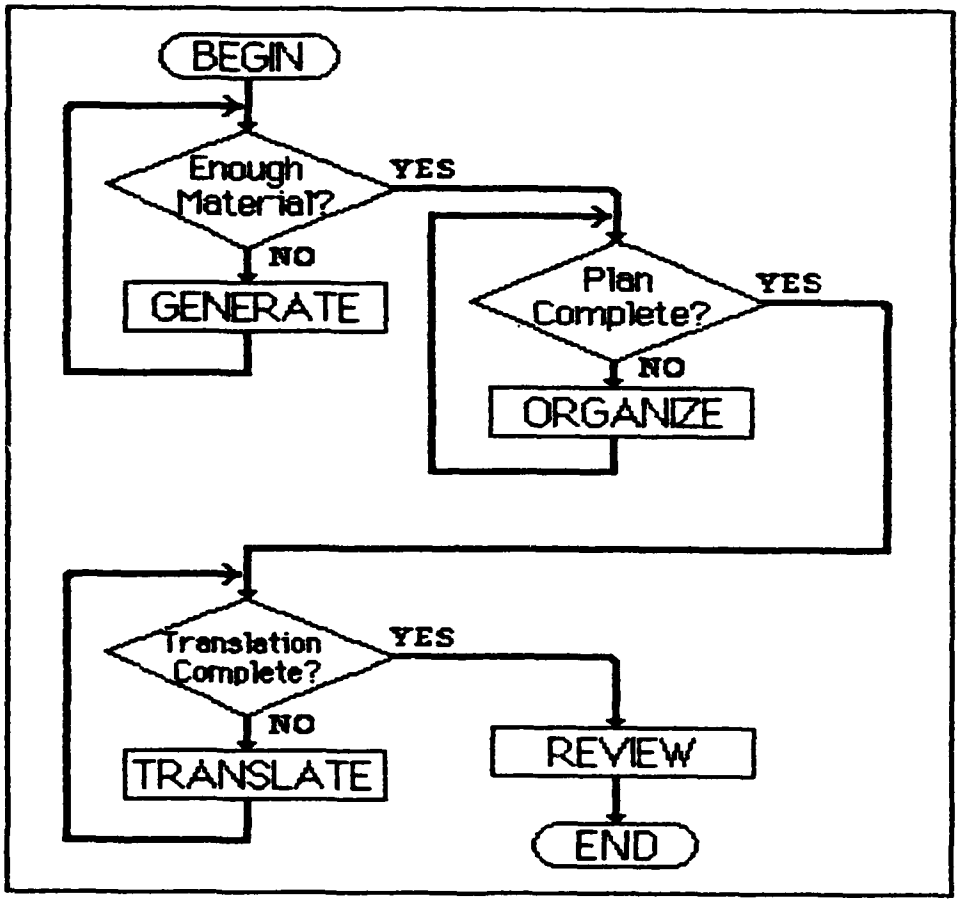


Figure 9. "Breadth First" Writing Style

Since all humans develop one of these four styles as they learn to communicate, an idea processor intended for widespread use must accommodate all of these styles. Huber argues strongly that a software package should not restrict a user in any way, but should allow that user to perform tasks using whatever style he or she brings to the situation. (17:575)

Requirement:	An idea processor must allow easy movement between different display modes and idea processing functions.

Summary of Requirements

Based on the relevant theories found in the current literature, the following requirements for an idea processor were derived. An electronic idea processor must

- support lists, sets, and hierarchies of ideas
- support sequential text to be associated with each idea
- allow easy insertion of new ideas
- allow easy manipulation of ideas, such that they can be selected, moved, inserted, changed, and removed
- support grouping (classification) and ordering (prioritization) of ideas
- allow easy insertion and editing of the textual information associated with a specific idea
- allow easy viewing of any and all ideas and their associated text

- allow the user to preview the text associated with ideas in an integrated, sequential manner (as it would appear in a document)

- allow easy movement between different display modes and idea processing functions.

IV. Design and Implementation

Design

Existing Software. In addition to information on relevant theories on composition, the literature also provided several articles on "outline generators" which satisfy some of the requirements for an idea processor. One of the very first outline generators was ThinkTank*, a program for the IBM PC that allowed the user to manipulate topics in a "table of contents" format. Several software packages followed ThinkTank's lead, including MaxThink*, KAMAS*, THOR*, IDEA!*, and Executive Writer/Executive Filer*. Supplemental software that adds idea processing capabilities has been made available for several existing word processors, such as Display Write 3* and Freestyle*. Integrated software packages such as Symphony* (with optional outlining package added) and Framework* merge idea processing with the more traditional word processing, data base management, and spreadsheet functions.

---

*ThinkTank is a trademark of Living Videotext Inc.; MaxThink is a trademark of MaxThink Inc.; KAMAS is a trademark of Kamasoft Inc.; THOR is a trademark of Fastware Inc.; IDEA! is a trademark of Traveling Software; Executive Writer/Executive Filer is a trademark of Paperback Software International; Display Write 3 is a trademark of International Business Machines Inc.; Freestyle is a trademark of Summa Technologies; Symphony is a trademark of Lotus Development Corporation; Framework is a trademark of Ashton-Tate Inc.

Initial Design. The general features of these software packages were evaluated with respect to the theories of composition identified in the literature review. This evaluation served as a departure point for designing the idea processor developed in this research project. The design was then modified to take into consideration the hardware currently in use in the Air Force.

The initial design that was developed in this research consists of a hierarchy of computer software routines and a basic set of data structures that together define the functions and capabilities of an idea processor.

Data Structures. A special double linked-list data structure is the primary means for storing the user's ideas in a hierarchy of lists. A linked-list is a list of items contained in a table, such that associated with each item are two "links" (pointers that refer to other items in the table): a link to the preceeding item in the list, and a link to the next item. The linked-list structure used in the initial design has a third kind of link that refers to subordinate items. This "doubly-linked" list can be viewed as having links in both horizontal (sequential lists) and vertical (depth of hierarchy) dimensions.

A visual representation of the linked-list data structure is shown in Figure 10. Each item in the linked-list has a separate variable for the name of the idea, and

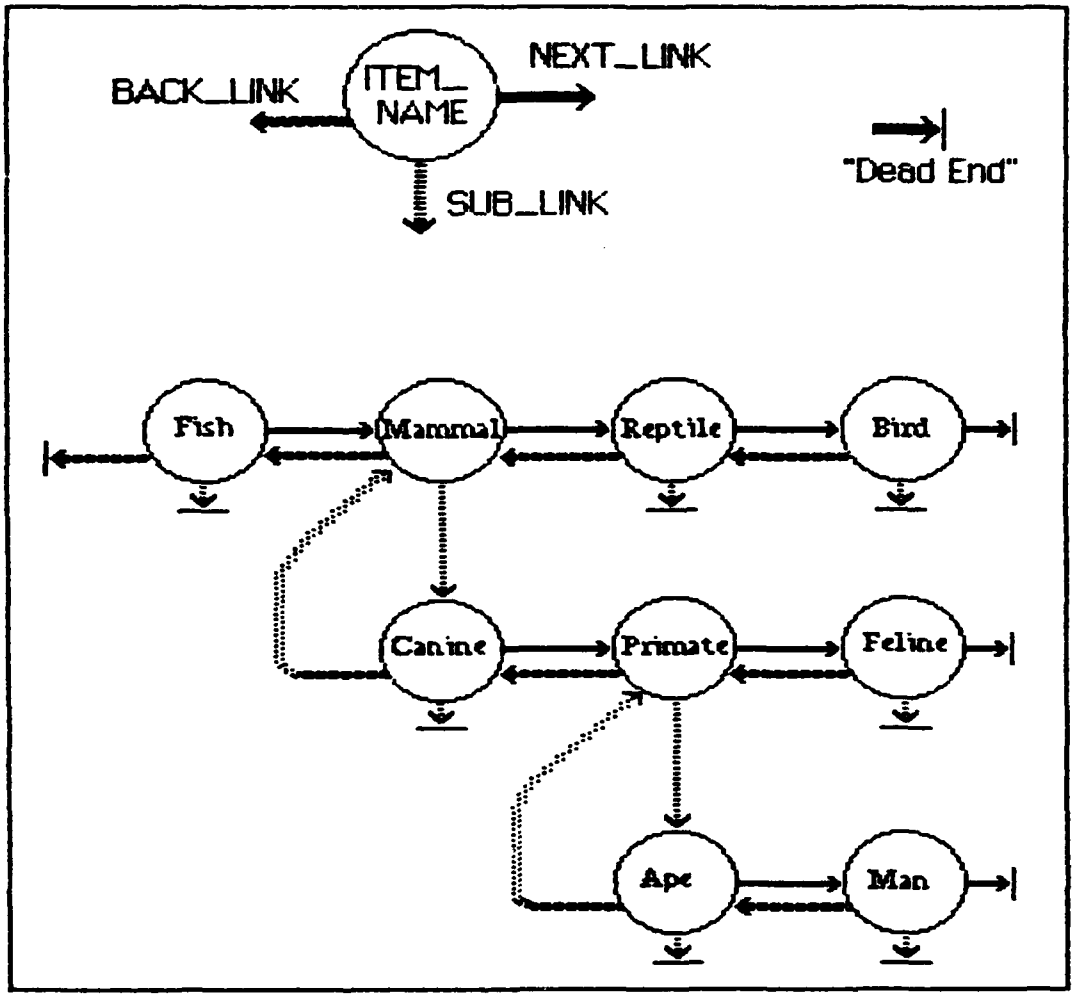


Figure 10. Linked-List Data Structure

pointers to the next, previous, and subordinate items in the hierarchy. A pointer value of zero represents a "dead end." For example, if a particular idea has a NEXT_LINK of zero then it is the last idea in a list at one level of the hierarchy. Table I lists the linked-list's components and an explanation of each.

TABLE I

Data Definitions for Initial Design

Linked List Definitions

LINKED_LIST	The primary data structure in the idea processor, used to store information entered by the user into a hierarchical format. It consists of a one-dimensional table of entries, where each entry includes the following fields:
ITEM_NAME	a string of characters representing the name of an idea or topic
ITEM_TEXT	the text associated with the idea
SELECTED	a flag indicating that the idea has been marked, such that it will be affected by any function that operates on a group of ideas
NEXT_LINK	an integer that refers to the next item in a list of items (equal level of importance)
BACK_LINK	a pointer to the entry that is pointing to this entry: the previous item in the list, or the "parent" item if this entry is the first in the list
FIRST	a boolean (TRUE or FALSE) variable that is true if and only if this entry is the first entry in a list; Indicates whether BACK_LINK points to its "parent" (if TRUE) or the previous item in a list (if FALSE)
SUB_LINK	a pointer to the first (or only) item in a list of items subordinate to this entry

NOTE: For all pointers (variables ending in _LINK), a null value (zero) represents no link, or a "dead end". For any one LINKED_LIST entry, a NEXT_LINK of zero indicates that it is the last entry in a list; a SUB_LINK of zero indicates that there are not yet any items subordinate to this one. However, since a LINKED_LIST entry must always have either a "parent" (if it is first in a list), or a previous item (if it is somewhere else in a list), then BACK_LINK should never be zero.

The linked-list structure is controlled by a set of working variables, defined in Table II. These variables keep track of the current idea being operated on (under the cursor), the current "depth" in the hierarchy, a "stack" that remembers the path of ideas that lead down the hierarchy to the current idea, and other parameters.

Program Structure. As shown in the structure chart in Figure 11, the main idea processor program is supported by five groups of subroutines which perform cursor movement, text editing, idea manipulation, display management, and file control. Most subroutines in this design correspond to specific functions, while others perform invisible tasks such as updating the display, allocating free variable space, receiving and checking input from the user, and maintaining the linked-list hierarchy.

Although this initial design fully specified most aspects of an idea processor, some of the more machine-dependent tasks were left intentionally unspecified. The display and text editing routines were only generally defined prior to implementation on an actual microcomputer.

Prototype Implementation

In order to validate the idea processor design derived from current theories of composition and existing software, a prototype was implemented to be tested in a validation

TABLE II

Data Definitions for Initial Design

Working Variables

TOP	A pointer to the "root" entry in the hierarchy (tree), the first item in a list of the highest ranking topics or ideas (chapters, sections, etc.)
CURRENT	A pointer to the current LINKED_LIST entry, the idea or topic that is now being operated on.
FREE	A pointer to the first LINKED_LIST entry that is not being used to store information. That entry, in turn, has its NEXT_LINK pointing to the second "free" entry, and so on, such that all unused LINKED_LIST entries are linked together.
LEVEL	An integer that represents the current "depth" in the hierarchy, or the number of "parents" that the current entry has.
STACK	A table of pointers such that each entry of the table points to that level's "parent" item.
SELECT	A list of pointers to all LINKED_LIST entries that have been selected (or marked) by the user. A zero entry represents the end of the SELECT list.

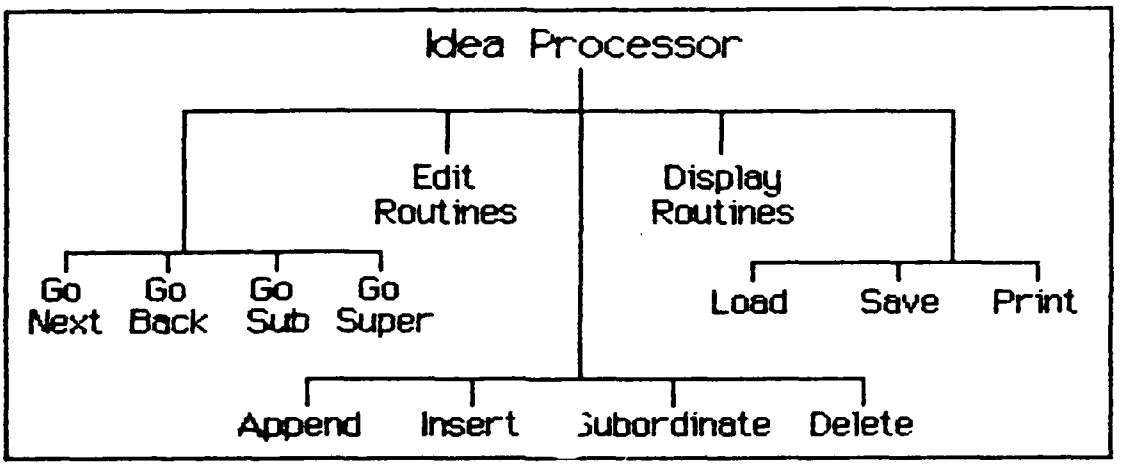


Figure 11. Idea Processor Structure Chart

experiment. The Burroughs B-20 series microcomputer system was selected as the target of the implementation for several reasons: (1) it was readily available at AFIT, (2) it was fairly familiar to the AFIT personnel who would serve as participants in the validation, and (3) it had several programming languages available for development of the prototype.

Preliminary Implementation. Before the idea processor design was fully complete, the BASIC programming language was used to explore the suitability of the Burroughs microcomputer for the prototype implementation. The BASIC language was chosen for its familiarity to the researcher and its simplicity for early development work. Several design approaches were explored and dropped as a result of insights learned while developing these early programs.

Final Implementation. Once the idea processor design was fully specified, the prototype idea processor to be used during validation was implemented in the Pascal programming language. The researcher chose the Pascal language for the following reasons:

(1) the researcher was very familiar with this language,

(2) the implementation of Pascal is fairly standard between computers, greatly simplifying the task of transferring the idea processor to other Air Force computers,

(3) as a compiled language, Pascal would execute much faster than an interpreted language like Basic, resulting in a reasonably fast idea processor,

(4) the structure inherent in Pascal would support the modular approach taken in the design,

(5) the ability to define new data types in Pascal would enhance the use of abstract concepts in the idea processor design, and

(6) the English-like style of Pascal would result in a very readable program.

Appendix A contains a source code listing of the prototype idea processor. The prototype implemented most aspects of the design, although some were excluded due to lack of time or programmer limitations.

Description of Prototype

Display. The prototype idea processor's screen has three distinct regions (shown in Figure 12), consisting of the idea processing window, the status line, and the text

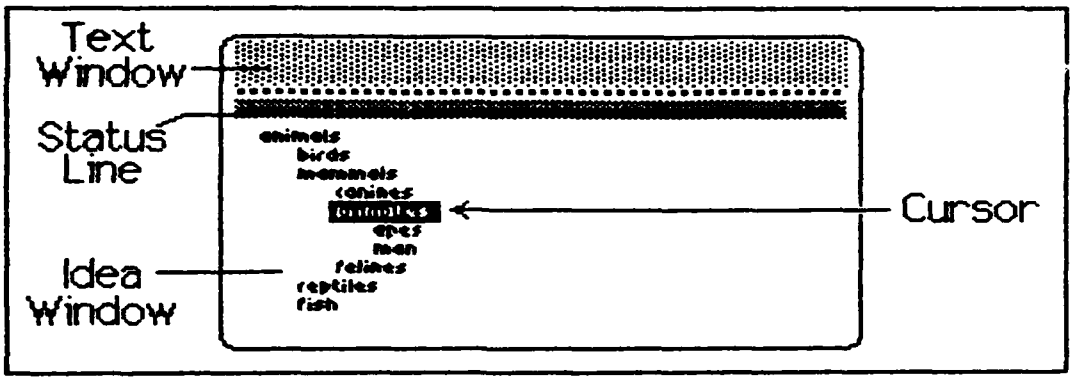


Figure 12. Prototype Idea Processor Display

window. The entire screen always represents the current state of the conceptual document, even if not all ideas or their associated text can be seen at all times.

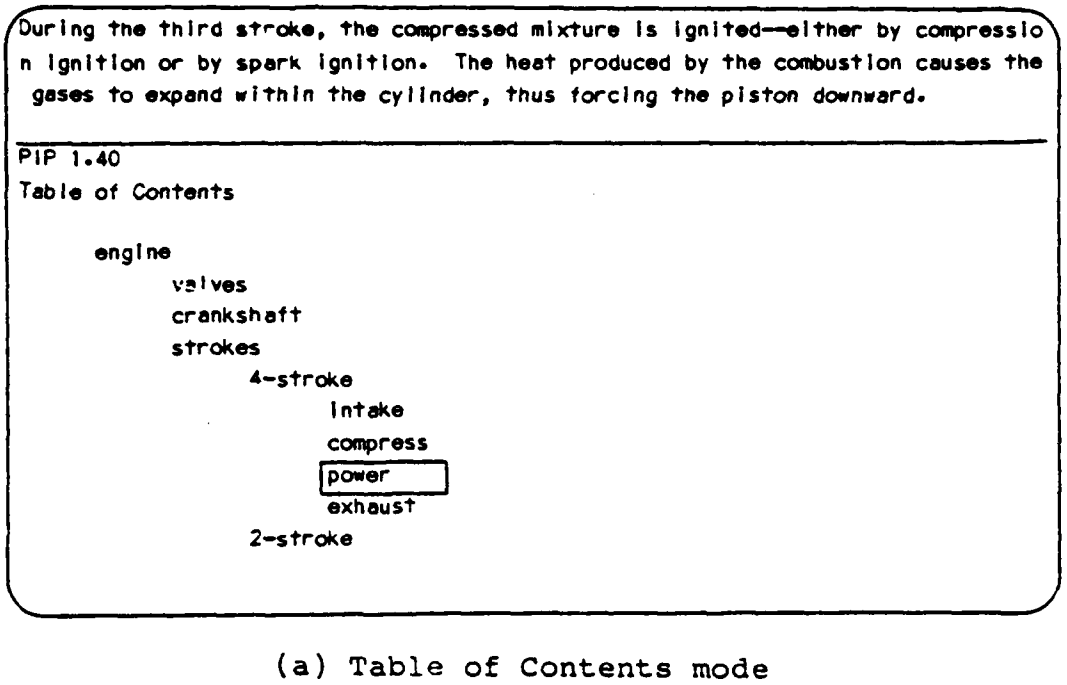
Idea Window. The primary display window on the idea processor screen is the area in which ideas are displayed as they are manipulated. Within this window, topics and subtopics are arranged to show their underlying hierarchical structure. One idea is always displayed in reversed video--this is the cursor, representing the "current" idea being operated on.

Display Modes. The topics and subtopics shown in the idea window can be displayed in either of two modes: the "Table of Contents" mode, in which ideas are listed sequentially down the screen, with each idea indented

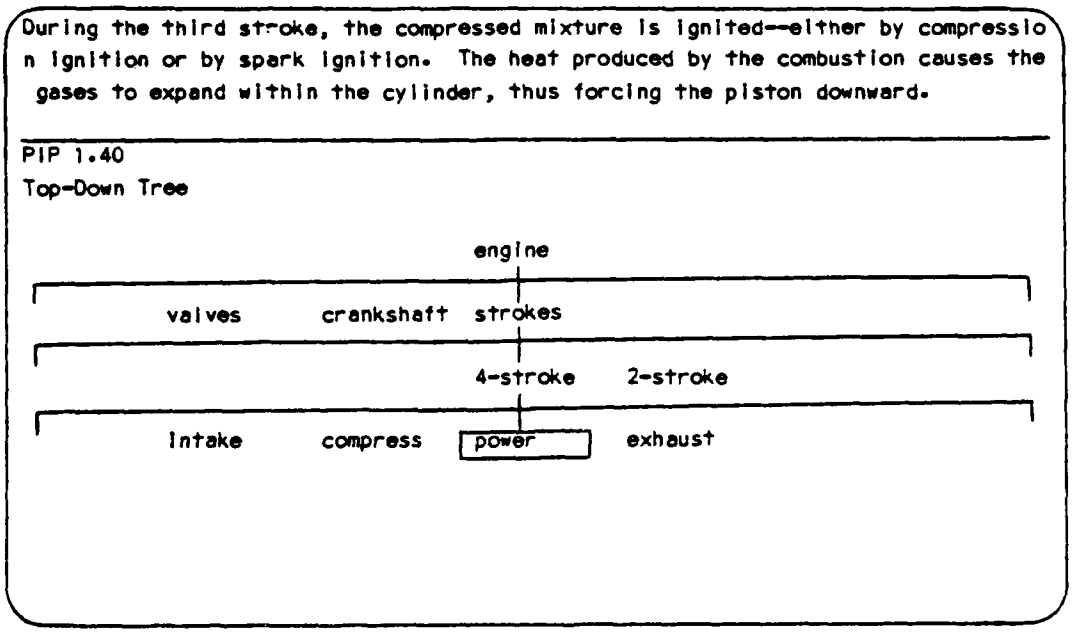
according to its "depth" in the hierarchy; or the "Top-Down Tree" mode, where the ideas are arranged in a vertical, upside-down tree, with the main topic at the top-center of the window and subordinate topics beneath it. Figure 13 shows example screens from both modes. The display mode can be changed at any time by pressing the CHANGE MODE (F1) function key. The current display mode does not in any way affect the actual structural relationship of ideas in the prototype.

Text Window. At the top of the screen three lines are reserved to display up to 240 characters of text associated with the current idea (the idea highlighted by the cursor). This is enough text to hold one very long sentence or several short ones. This window is separated from the rest of the screen by a fourth line. Words are not automatically wrapped at the margin within the text window, but are always broken at the 80th column on the screen.

Status Line. Located between the idea and text windows is one line of information about the current state of the prototype idea processor. The software version number and current display mode are shown at the left end of the line, and any new idea being added is displayed in the middle of the line. When the load, save, or print functions are selected, the name of a file is displayed in the center of the status line as it is entered by the user.



(a) Table of Contents mode



(b) Top-Down Tree mode

Figure 13. Example Displays from Prototype Idea Processor

Movement Functions. Movement of the cursor from one idea to another is accomplished by use of the cursor "arrow" keys. As shown in Figure 14, these keys are found in the upper right corner of the Burroughs keyboard. The direction of movement that results from a cursor keypress depends on the current display mode, as shown in Table III. Although the functions of the cursor keys appear to vary drastically for different display modes, they are actually consistent with the structure of the ideas on the screen at any time.

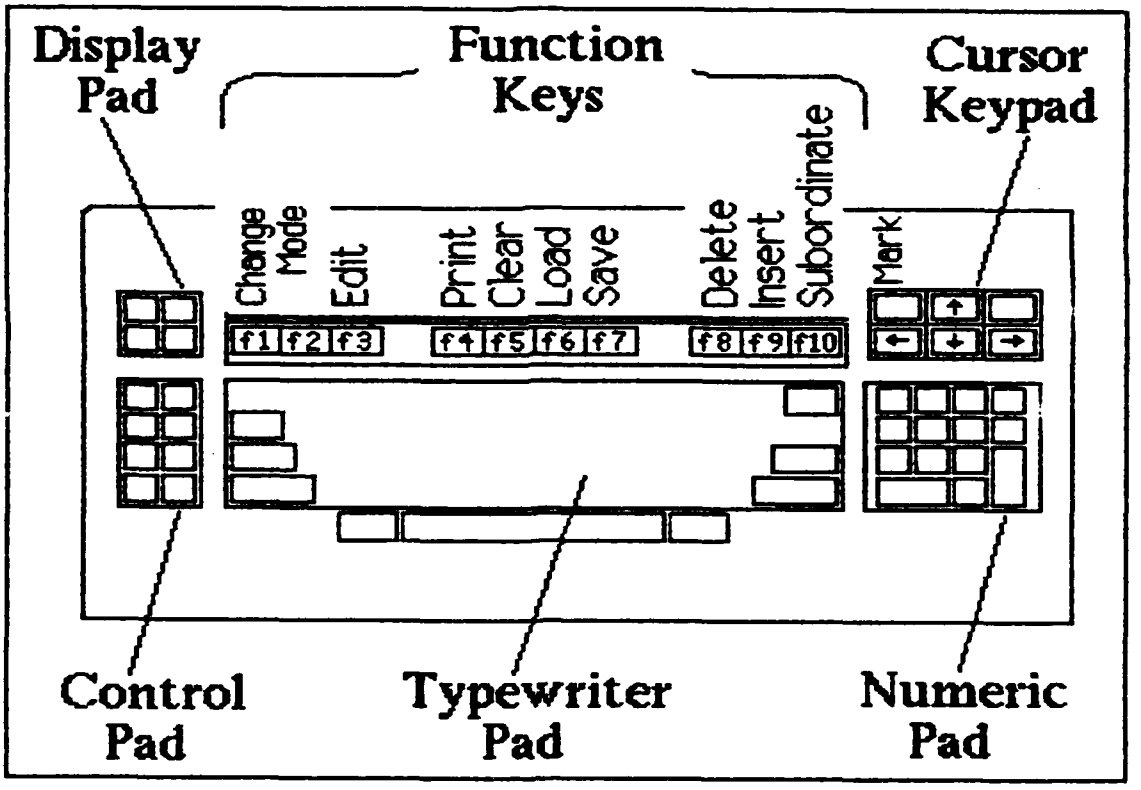


Figure 14. Burroughs Keyboard (3:Chap 20,2)

Basic Manipulation Functions. Several fundamental idea processing activities, identified as requirements in

TABLE III

Cursor Key Functions

CURSOR KEY	DISPLAY MODE	
	Table of Contents	Top-Down Tree
Up Arrow	go to previous	go to superordinate
Down Arrow	go to next	go to subordinate
Right Arrow	go to subordinate	go to next
Left Arrow	go to superordinate	go to preceeding

Chapter III, are supported by the prototype. New ideas may be added before or after ideas in a list, or as a subtopic to an existing idea. Any idea may be deleted, unless it still has subtopics beneath it.

Append. If the user begins typing new ideas into the prototype without specifying where they are to be added, the prototype assumes that the ideas are to be appended to the current idea (the idea highlighted by the cursor). For this reason there is not a specific APPEND key. As the user types the name of the new idea, the characters appear in center of the status line at a flashing underline marker. The user can correct mistakes using the BACKSPACE key, but must press the RETURN key when the new idea name is complete. Once the new idea name has been entered, it

disappears from the status line and appears at the appropriate place in the idea window.

Insert. It is frequently desirable for a new idea to be added in front of another in a list of ideas--the insert function is used for this purpose. After pressing the INSERT (F9) function key, the user enters the new idea name in the same manner as for the append function.

In most cases either the append or insert function could be used, but two special cases exist that make it necessary to have both: to add an idea to the beginning of a list of topics, the insert function must be used; and to add an idea to the end of a list, the append function is required.

Subordinate. When a new idea is subordinate to an existing idea, the new idea is added as a subtopic with the SUBORDINATE (F10) function key. Once the name of the new subtopic has been added in the same way as for the append and insert functions, the highlighted idea cursor stays on this new lower level of the hierarchy until the user specifically moves back to the "parent" topic. More subtopics can be appended after the first subtopic, or inserted. If even lower levels of ideas are needed, "sub-subtopics" can be added by again using the subordinate function.

Delete. When an idea needs to be removed from the hierarchy, the user moves the cursor to that idea and presses either the DELETE (F8) function key or the special DELETE key at the left edge of the Burroughs keyboard. The current idea is removed from the list, such that no "hole" is left between the two ideas on either side of the deleted idea. The ultimate position of the cursor depends on the position of the deleted idea, but the cursor always moves the most logical "next" idea in a list, or to the superordinate idea if the deleted idea was the only subtopic under a topic.

Selection Functions. While most idea processing is performed on one topic at a time, it is sometimes useful to perform the same function on several ideas at once. The prototype idea processor supports the selection and manipulation of a list of specific ideas by the use of the MARK key, found in the upper right-hand corner of the Burroughs keyboard. When the MARK key is pressed, the idea under the cursor becomes "selected", or put in a list of ideas to which the next manipulation function will apply. A selected idea is "deselected" by moving the cursor over it and pressing the MARK key again. Any number of ideas can be selected in this manner, and in any order.

Whenever at least one idea is on the "selected" list, the red lights on the INSERT, SUBORDINATE, and DELETE

function keys are turned on to remind the user that those functions will operate on the selected ideas, not the idea under the cursor. The prototype remembers the exact order in which the user selected ideas and performs the manipulative function to each idea in that order.

Mark-Insert. The user can move ideas within the hierarchical structure by selecting any number of ideas, moving the cursor to the idea he or she wants the other ideas to be moved in front of, and pressing the INSERT (F9) function key. Since the prototype remembers the order in which ideas are selected, this function can be used to change the order of a list of ideas.

Mark-Subordinate. If the user wants to make several ideas become subtopics to a main topic, he or she can select the sub-ideas, move the cursor to the main idea, and press the SUBORDINATE (F10) function key. As with the mark-insert function, the sub-ideas can be selected in the exact order that the user wants them to be in the list of subtopics. This allows the user to perform both grouping and ordering on ideas at the same time.

Mark-Delete. Many ideas can be removed at the same time through the use of the mark-delete function. The user marks the ideas to be deleted and presses the DELETE key. As with the basic delete function, an idea will not be deleted if it still has one or more subtopics beneath it.

File Control Functions. The prototype idea processor uses a special file format to keep the underlying structure of a document while it is processed. This structure, including all ideas and associated text, can be saved to disk or loaded from disk, but it is the user's responsibility to name and manage the files. Unlike other Burroughs applications (such as WRITEone*), which support the concept of named documents using "open" and "close" functions, the prototype does not keep track of what document it is processing, nor does it automatically save a document before loading a new one or exiting.

Load. Existing idea processing documents are loaded using the LOAD (F6) function key. The prototype prompts the user for the name of a file. Once the user enters the file name, the prototype attempts to open this file and read the ideas and text into the Burroughs memory. The prototype does not warn the user to save the document that was in memory before the load function was initiated.

Save. When the user wishes to save the ideas and text of a document, he or she presses the SAVE (F7) function key. The user is prompted for a file name, and when it is supplied, the prototype looks for the specified file. If it exists, the new document is overwritten on top of whatever

---

*WRITEone is a trademark of the Burroughs Corporation

was in the file. If the specified file does not exist, a new file is created and the document is saved in it.

Print. Since the ultimate form of a document created on the prototype idea processor should be some format that could be edited on a standard word processor, a means must be provided to translate the structure of ideas and text into a sequential text file. This is accomplished using the print function, which does not actually cause the document to be printed, but creates a text file compatible with word processors.

When the user presses the PRINT (F4) function key, he or she is given two choices for the format of the text file: "Paragraph" format, which merges all text from all ideas into a continuous stream of sentences; or "Bullets" format, which creates separate lines of text, each indented and preceded by dashes as in a "talking paper" format. The text file created in either format by the print function can be opened by a word processor, edited, and printed as a formatted, word-wrapped document.

Other General Functions

Edit. When the user wishes to associate text with a specific idea, he or she moves the cursor to that idea and presses the EDIT (F3) function key. The prototype then allows the user to enter up to 240 characters in the three lines at the time of the screen. Words are not

automatically wrapped at the margins. Once the RETURN key is pressed, the text is entered "as is" into the structure of the document. If the user presses the EDIT key for that idea again, the associated text is cleared and must be typed again.

Change Mode. The user can change the way in which the hierarchy of ideas is organized on the screen by pressing the CHANGE MODE (F1) function key. This function switches the display mode between the Table of Contents and Top-Down Tree modes, with no change to the actual ideas and text currently in the idea processor.

Clear. If at any time the user wishes to remove the ideas and text from the idea processor, he or she can press the CLEAR (F5) function key. The idea window will then show the resulting empty hierarchy. It is the user's responsibility to first use the SAVE (F7) function key to save the idea processor document for later use.

Finish. Once the user is done processing ideas, and has saved any information that might be needed later, he or she simply presses the FINISH key in the bottom left corner of the Burroughs keyboard. After a few seconds the Burroughs operating system will return a command line.

Limitations. As implemented in this research, the prototype idea processor imposes several limitations that the initial design did not have. The total number of ideas

that can be processed is 25, while the number of ideas that can be displayed in Table of Contents mode is 18. This last limitation is caused by a lack of scrolling or paging to accomodate more than one screen of ideas. The maximum number of levels in the hierarchy that can be supported is ten.

The prototype provides little or no error checking or error recovery. There is no way to "abort" the entry of a new idea name--the user must eventually press RETURN once he has started typing the name at the center of the status line. The prototype does not filter the characters typed by the user to prevent "non-printing" characters that can cause the software to behave erratically or halt abruptly. When receiving a file name from the user for the load function, the prototype has no way to recover if that file does not actually exist. If the file specified in the save and print functions already exists, the prototype overwrites the file without warning the user.

V. Validation and Analysis of Results

Validation

Purpose. The purpose of the validation experiment was to validate the initial design developed in this research and implemented in the prototype. Specifically, this validation sought to confirm that:

(1) an idea processor must meet certain requirements in order to allow a writer to effectively manipulate and prepare his or her ideas for communication,

(2) the design developed in this research met those requirements, and

(3) the prototype correctly--if not completely--implemented the design.

Location. The validation was conducted at the Air Force Institute of Technology School of Systems and Logistics (AFIT/LS), in the offices of the Department of Research and Communication (LSH). The validation was conducted on the weekend of 16-18 May, 1986, a time when the office was quiet and without distractions.

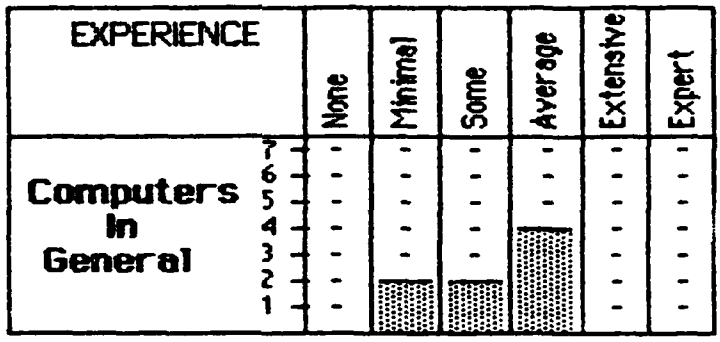
Participants. Eight AFIT graduate students participated voluntarily in the validation. These students were identified by their communication instructors as performing exceptionally well, and so were considered "experts" for the purposes of the validation. The eight validation participants were:

Captain Robert A. Eaton, USAF
Captain Michael R. Fredette, USAF
Captain Richard J. Ingenloff, USAF
Captain George S. Maxwell III, USAF
Captain Judyann L. Munley, USAF
Captain Keith E. Smith, USAF
Captain Franklin L. Williams, USAF
Captain Adelle R. Zavada, USAF

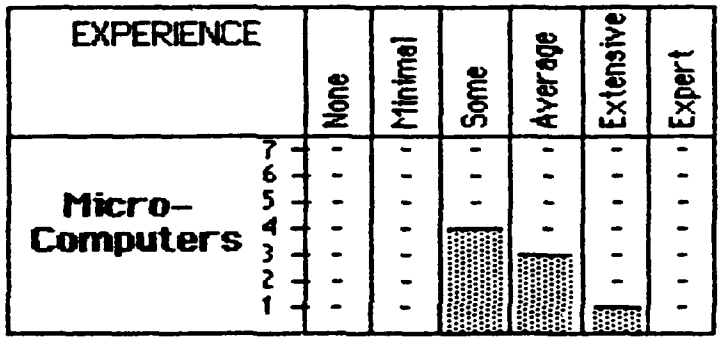
Before the validation, each participant was asked to complete a pre-validation survey, distributed to them one week before the scheduled sessions. This survey gathered general information on each participant's previous experience with computers and various types of software. It also asked about the participant's general approach to composing a writing assignment. Some of the results of this survey are shown in Figure 15. Table IV provides general demographic information about the participants.

Equipment. Each participant used a Burroughs B-29 microcomputer system with CPU, monitor, keyboard, and floppy disk drive. Participants did not have access to any other materials.

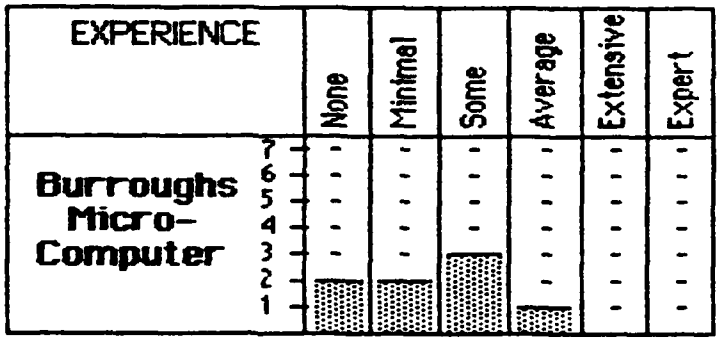
Software. The Burroughs B-29 system was running version 1.40 of the prototype idea processor at all times during the validation, except that the first participant used version 1.39. The changes made to version 1.39 to upgrade it to version 1.40 included the addition of structure lines to the "Top-Down Tree" mode, and the removal of several obvious program errors. The program was



(a) Experience with Computers in General

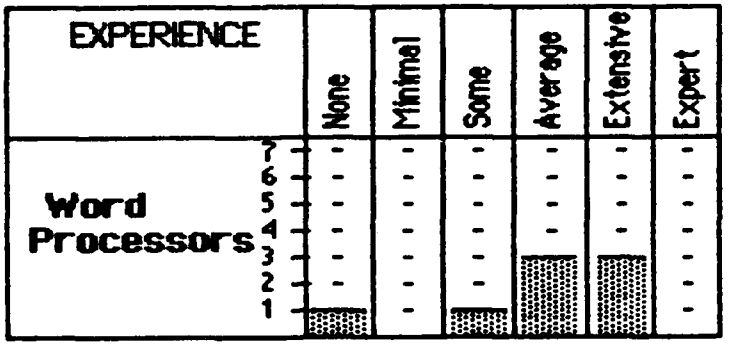


(b) Experience with Microcomputers

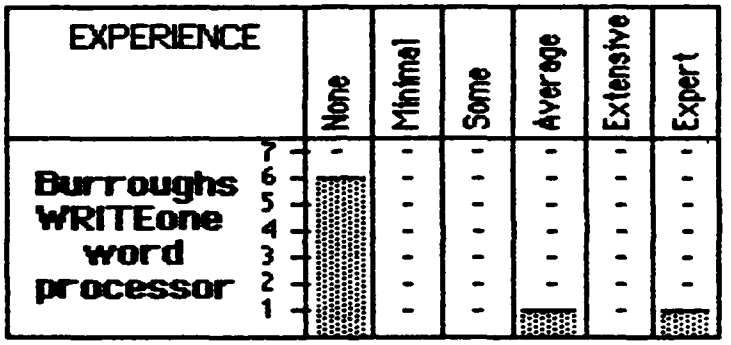


(c) Experience with Burroughs Microcomputers

Figure 15. Participants' Responses to Pre-Validation Survey



(d) Experience with Word Processors



(e) Experience with WRITEone*

Figure 15 (continued). Participants' Responses to Pre-Validation Survey

*WRITEone is a trademark of the Burroughs Corporation

TABLE IV

Participant Demographics

SEX		DEGREE PROGRAM	
male	6	Engineering Mgt	3
		Maintenance Mgt	2
female	2	Logistics Mgt	1
		Transportation Mgt	1
		Systems Mgt	1

installed on the hard disk shared by all terminals in the LSH and LSM offices.

Schedule. Each participant was scheduled for an individual two-hour session. There was minimal pressure on the participants due to time constraints: all eight participants completed the validation in less than the allotted time.

Tasks. Two idea processing tasks were generated from encyclopedia articles (5; 18). Separate sentences from each article were scrambled to produce unstructured notes similiar to those that would be taken during research. Appendix B lists the text of the idea processing tasks.

Procedures

Arrival. As each participant arrived, he or she was greeted and given a few minutes to relax and get comfortable. The pre-validation survey was collected. Two participants had forgotten to bring the pre-validation survey, and so were given time to complete another one.

Orientation. The researcher asked the participant to sit at the B-29 microcomputer workstation. Using an outline, the researcher explained to the participant what activities were to take place during the validation. This orientation included the philosophy of the idea processor, the general purposes of the validation, and what would be expected of the participant.

Training. The researcher then presented an informal explanation of the prototype idea processor. As each feature was described, the researcher demonstrated on the actual prototype running on the B-29. The pace of this training session was determined by each participant; only when he or she understood one function did the researcher move on to the next.

Practice. Once the operation of the prototype idea processor was presented, the participant was allowed to practice using the prototype for as long as he or she wanted. The researcher continued to answer any questions on the use of the idea processor. A practice file was used during both the training and practice periods; the information in this file was very similar to those used in the writing tasks.

First Task. Once the participant indicated that he or she felt ready, the researcher brought a new set of information into the prototype idea processor. The idea

processing task consisted of using the idea processor to reorganize this information into a small document. The participant was given as much time as needed to accomplish this task, and was allowed to ask questions about functions of the idea processor, but the researcher did not help the participant accomplish the idea processing task.

Recording. As the participant accomplished the writing task, he or she was encouraged to talk out loud about what he or she was trying to do. Participants were asked to verbalize any obstacles they encountered in trying to solve the writing tasks by their customary styles of organization. The researcher made written records of the participants' comments and his own observations.

Second Task. Each participant worked until he or she felt satisfied with the organization of the writing task information. Since each participant was considered an "expert" communicator, no evaluation was made of the final organization. Participants then accomplished a second writing task in the same manner as the first. This second task gave the participant more time using the prototype, such that any limits to performance during the first task that were caused by incomplete learning could be improved on during the second task. All eight participants used the same practice file and the same two writing task files.

Discussion. After completing the second writing task, each participant was asked to discuss his or her overall perception of the usefulness of the prototype idea processor. All participants were willing to express both general impressions and specific suggestions for improvement. Some participants asked questions about how the validation was going and where the research was expected to go in the future.

Analysis

Classification Criteria. Appendix C lists the obstacles recorded by the researcher during the validation, prior to their being separated into separate classes. Each obstacle that a participant encountered during the validation could be classified as one of the following types: requirements obstacle (type 1), design obstacle (type 2), implementation obstacle (type 3), and environment/personal obstacle (type 0). Each type of obstacle represents a different potential source of difficulty associated with the prototype idea processor.

In classifying obstacles from the recorded notes of the validations, the researcher first looked at an obstacle with respect to the requirements. If the ultimate cause of a participant's obstacle was an incorrect or missing requirement, it was classified as type 1. If a correct

requirement existed, but was not reflected in the design, the obstacle was classified as type 2. If an obstacle was caused by an incorrect implementation of a requirement and design feature, then the obstacle was classified as type 3. If the source of an obstacle was found to be the design of the validation experiment, the environment in which the validation was performed, or with the participant's own writing ability, then the obstacle was classified as type 0.

These classified composition obstacles were collected from all participants and combined into a complete list of obstacles, grouped by type. Identical or very similiar obstacles were combined, so that a count of the number of observed occurances of each obstacle was recorded. Obstacles identified by two or more participants were considered for potential changes to the requirements and design.

Type 1 Obstacles. Three participants identified a human short-term memory limitation that forced them to constantly go back to key ideas and reread associated text, especially to check on the order of subtopics for a list. These participants expressed a need for some way to keep specified text available on the screen while moving or manipulating other ideas.

The idea of helping the communicator recall information was not addressed in the original requirements. Research in

psychology has investigated several aspects of short-term memory in human beings. Short-term memory is believed to be able to hold about seven (plus or minus two) bits of information for up to about 30 seconds (13:103). Other studies of information processing load suggest that the number of mental "schemes" that someone can keep track of in his or her mind grows from about two in early school years to about seven (2:81). The effects of this missing requirement will be discussed further in Chapter VI.

Type 2 Obstacles. The participants in the validation identified many deficiencies in the initial idea processor design. These obstacles range from general deficiencies to specific features that were missing or incorrectly designed.

Superordinate. Four participants identified the need for a function to "unsubordinate" or "superordinate" ideas--that is, to raise one or more ideas from a subtopic to a topic, or to elevate those ideas above the level that they were on. Six participants began each writing task by identifying the "top" or overall idea, moving that idea to the first in the list of all ideas, and then subordinating all other ideas to it. A superordinate function would allow writers to specify the main topic idea and raise it to the top of the hierarchy in much the same way that they do when they organize ideas in their mind.

Sequential Step. Four participants felt the need for a method of moving through the hierarchy of ideas in the sequential order of a document. These participants found it inconvenient to move down through a tree's subtopics and then have to move back up before going down the next "branch" (chapter, section, etc.) of the tree.

Undo. Three participants found the lack of an "undo" feature disturbing. This feature would have allowed the user to take his or her last action back, such that the idea processor would be in the same state as before that action was taken. Most users are more relaxed and spontaneous in their use of a computer when they have an "undo" key available to correct unintentional actions (9:37-38). A relaxed and uninhibited attitude is especially critical during the early stages of composition, when brainstorming and generation would be restricted by concern over accidental loss of information.

Type 3 Obstacles. Many of the obstacles that participants encountered during the validation were caused by errors in the implementation of the prototype. There were simple "bugs," inconsistencies between functions, confusing features, and arbitrary limitations. Some of these implementation obstacles were the result of limitations of the researcher's ability to program, as well as the limited amount of time available within the 15-month

thesis cycle. However, all of the following obstacles could be removed in an improved version of the prototype idea processor.

Mark-Append. Four participants found that there was no direct way to append more than one idea to an already existing list of ideas. Since there was no append function key, users were forced to either mark-insert or mark-subordinate.

Mark-Subordinate. Three participants discovered that mark-subordinate did not append more ideas to an existing list of subtopics, but instead replaced the old subtopics completely. The participants expected the new ideas to be added to the old ideas.

Mark-Delete. Three participants identified a obstacle when trying to delete several ideas where one of those ideas was the current idea (under the cursor). Where they expected all selected ideas to be deleted, the current idea was not deleted.

Abort Input. Four participants had serious trouble with the method used to add new ideas. The prototype idea processor required the users to first identify which function was to be performed (append, insert, or subordinate), and then to enter the name of the idea to be added. If no function was specified before typing the new idea's name, the idea processor assumed that the idea

would be appended to the current idea. However, as these four participants started to type the new idea name, they would suddenly decide where they wanted it to go (after, before, or subordinate to the current idea). The prototype did not provide a means by which users could stop adding a new idea, or allow the user to decide after the fact where to add the idea.

Bugs. Several errors in the implementation of the prototype caused repeated obstacles during the validation. These "bugs" generally caused incorrect or extraneous information to be displayed. Five participants discovered a bug that resulted in a tree's sub-sub-topics not being erased when the cursor was moved back to the top of the tree. Four participants found a bug that displayed "CURRENT =" and a number on top of the ideas on the screen; this information was used by the researcher during the programming of the prototype, but was mistakenly left in the validation versions. Another bug erased the status line information for two participants. Finally, two participants discovered that under certain conditions, an erroneous link between a sub-sub-topic and the main topic (top of the tree) allowed them to move the cursor in a continuous loop.

Text Area. The idea processor design was intentionally vague about how much text could be associated with each idea. While advanced implementation techniques

would have allowed almost unlimited text, the researcher had used a simple method that limited the total amount of text that the prototype idea processor could handle. This limitation, in turn, required a trade-off between the number of ideas and the amount of text associated with each idea. The balance selected for the prototype idea processor was 50 ideas, each with three 80 column lines of text, or approximately 240 characters per idea, which was considered enough to handle most long sentences, or several short, related sentences.

The validation participants all had different ideas about how much text was enough, but most agreed that more than three lines were needed. Three participants stated that five or six lines (400-480 characters) would be sufficient; several referred to the size of index card used for making research notes as being about this much. Two participants asked for enough text to contain a paragraph: at least half of the screen and preferably a full screen of text.

Tree-mode Cursor. Three participants were confused by the way in which the cursor appeared to move when using the "Top-Down Tree" display mode. Rather than moving in the direction of the arrow key pressed, the cursor (the idea displayed in reverse video on the screen) stayed in the center of the screen and the ideas shifted on the

line representing the list of ideas at the current level of topics. The participants saw the direction of movement of the ideas under the cursor as the direction of movement, whereas the actual direction of movement was just the opposite.

Type 0 Obstacles. There were relatively few obstacles that were clearly not caused by the requirements, design, or implementation of the prototype idea processor. Of the few obstacles that were not classified into types 1, 2, or 3, none was identified for more than one participant. This fact suggests that these type 0 obstacles were related to individual differences--rather than to limitations in the environment of the validation.

One obstacle that was clearly caused by the design of the experiment was the desire by one participant to completely reword the text of the document, rather than use the idea processor to reorganize the existing text. This participant stated that although he could use the prototype idea processor to perform the writing tasks, in actuality he would have preferred to start over and rewrite the documents. In this sense, the design of the validation experiment may have restricted all of the participants' abilities to retranslate ideas--as an alternative to reorganizing them.

Design Modifications

The initial design derived from current theory and existing software was modified based on the type 2 obstacles--deficiencies in the initial idea processor design--that were found in the validation. These design changes in no way reflect deficiencies in the original requirements as identified in the literature review.

Superordinate. A new function was added to the idea processor design to allow users to elevate specified ideas to a hierarchical level above the current one. This "superordinate" function yields the opposite result of the subordinate function. An idea processor with both of these complementary functions could be easily used by writers with either major approach to writing: integrators (who build groups of topics from the "bottom-up") and analyzers (who work "top-down" from the general to the specific).

Sequential Step. A relatively simple change to the initial design was made to allow idea processor users to move the cursor directly from the last idea of one "branch" of topics to the first idea of the next "branch". An idea processor with this change better reflects the way people "browse" through a document during the reviewing phase of composition.

Undo. An "undo" command was added to the idea processor design to meet the need that participants

expressed for some way to recover from a mistake. This feature allows the user of an idea processor to return a document to the state it was in before the last function was performed. In addition, the initial design was modified to include a list of "discarded" ideas that would be maintained out of sight of the user. Specifically, whenever the user selected and deleted one or more ideas, those ideas would not be permanently lost, but instead would be moved to an "invisible" list of ideas. This "discard" list could later be accessed, and selected ideas could be recovered and reincorporated in the document. This concept of never throwing any idea away more closely supports the principles used in brainstorming, such as not judging any idea while generating new ones.

VI. Conclusions and Recommendations

Conclusions

Requirements. During the validation, only one type 1 obstacle was identified: human beings are subject to short-term memory limitations, and an idea processor should take this into account and provide memory assistance to users during composition. This could be done by allowing the user to "freeze" some information on the screen while he or she continues processing other ideas, or by providing a "notepad" that the user can reference as needed. Other than this omission, the requirements determined from the literature review appear to be valid within the scope of this research.

Design. The final idea processor design, as modified based on the type 2 obstacles found in the validation, is described in Figure 16. This design represents the combined influences of current theories and literature, existing software packages, and the capabilities of microcomputers currently in use by Air Force personnel. This final design was found to be valid within the scope of this research.

Implementation. The prototype idea processor used in the validation represents a first step towards a more complete implementation of the design investigated in this

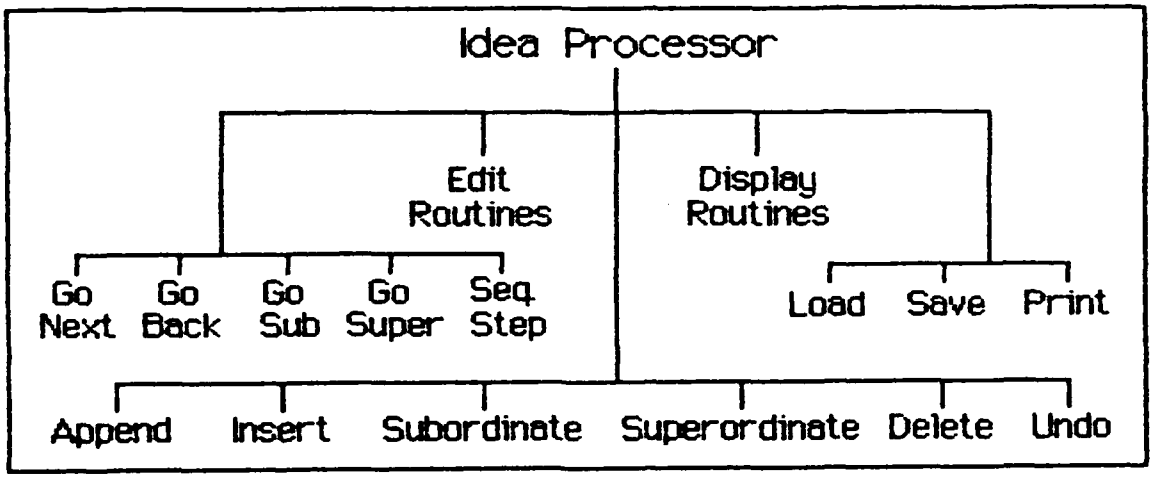


Figure 16. Modified Idea Processor Design

research. Many specific implementation problems were identified in the validation, but these problems would be easily corrected in an improved prototype. In addition, many opportunities for enhancements exist using this research's prototype idea processor as a starting point.

Recommendations

Improvements to Prototype. While the prototype developed during this research would serve as a minimal idea processor, it would more appropriately serve as the basis for a series of modifications and enhancements that would result in software closer to the ultimate goal of "computer assisted composition." These modifications would begin with the correction of errors, bugs, and other type 3 obstacles

identified in the validation. The idea processor should then be updated to reflect the improved final design shown in Figure 16. Finally, four opportunities for enhancements should be investigated: (1) correction of implementation errors, (2) improvement of the design, (3) removal of artifical limitations, and (4) incorporation of enhancements (as suggested by the validation participants).

Error Correction. A number of type 3 obstacles identified in the validation should be corrected in the original prototype idea processor. These errors include anomalies in the operation of the mark-append, mark-subordinate, and mark-delete functions; a confusing method of idea input that does not allow cancellation; a limited amount of text that can be associated with each idea; and a number of specific display aberrations. Many of these obstacles have already been corrected in a post-validation version of the prototype.

Design Changes. The final idea processor design reflects changes made to the initial design based on type 2 obstacles found during the validation. These changes should be incorporated in the prototype to result in a complete and correct implementation of the final idea processor design. This would require the changes to the prototype to add a superordinate function, the ability to step sequentially through a document's ideas, and an undo function.

Limitations. Several limitations on the prototype idea processor were necessary due to the scope of the research, but could be removed in an improved idea processor. The maximum number of ideas supported could be raised from 25 to several hundred. This would require some method of scrolling or paging between multiple screens of ideas. The length of the string of characters given to an idea's name could be expanded from ten characters to the width of the screen. The amount of text associated with each idea could be increased to a full screen or more, with the text portion of an idea displayed in a window (size selectable by the user). Full text-editing features could be added to the idea processor, or existing word processors could be fully integrated with the idea processor.

Enhancements. The validation participants identified many potential improvements to the prototype idea processor based on their own experiences in composing communication. While many of these proposed enhancements were not classified as type 1, 2, or 3 obstacles because they did not meet the criteria of at least two people identifying the same obstacle, they are still insightful and useful ideas. Some of these enhancements include:

- changing the file concept from "load/save" to "open/close", making the idea processor more similar to many existing word processing packages;

- the functions could be activated by any combination of function keys, control-keys, escape-key sequences, command words, or menu selection, possibly including user-definable function names and keys;

- a search or modify function to find specific ideas or phrases in text;

- automatic numbering of ideas according to any one of several numbering schemes (I. A. 1. a., 1.2.3.4, bullets, etc.);

- relational data base features, allowing storing and searching for special "fields" of information (topic, source, author, citation);

- a help function to provide assistance to the user on any one of several levels;

- specific functions to move the cursor to the TOP or BOTTOM of the current list, UNMARK a previously selected idea, BOUND a range of ideas for selection, SWAP or SWITCH the order of several ideas.

Further Research. The research described in this thesis was exploratory in nature: it represents a first step in developing an idea processor for use on Air Force computers. Further research on the design, selection, and application of Idea Processors to Air Force use is desirable. More complete validation of the requirements and design in a more detailed scientific experiment could be accomplished using an improved and enhanced prototype idea processor. Existing software could be evaluated in a manner similar to the validation used in this study. Ultimately, operational field testing of a "production" quality idea

processor by Air Force personnel (with a variety of job responsibilities) should be accomplished. Future research could also investigate the use of a graphics interface (like that used by the Apple MacIntosh) for idea processing.

Summary

The purpose of this research was to determine a set of requirements based on current theories of composition, and to develop an idea design for such an Idea Processor, optimized for Air Force standard microcomputer systems. The basic concept of computer support during the early stages of composition has been shown to be feasible in this exploratory research. The resulting requirements and design appear to be valid within the scope of this original objective, and provide a starting point for more extensive research to be conducted in the future.

APPENDIX A

PASCAL SOURCE CODE FOR

PROTOTYPE IDEA PROCESSOR

```
< $LINESIZE:79 >
< $SYMTAB- >
< $TITLE:'Prototype Idea Processor' >
< $SUBTITLE:'     Version 1.40' >
< $MESSAGE:'compiling PIP 1.40' >

program PIP (INPUT,OUTPUT) ;

<*************************************************************
*                                                            *
*                 PROTOTYPE  IDEA  PROCESSOR                 *
*                                                            *
*           Conceived, designed, and implemented by          *
*                         ANDY MOORE                         *
*                                                            *
*                    as part of a thesis for                 *
*             the Air Force Institute of Technology          *
*                School of Systems and Logistics             *
*              Graduate Systems Management program           *
*                                                            *
*------------------------------------------------------------*
*                                                            *
*  The purpose of this program is to support the primary     *
*  goal of my thesis, which is to validate a hypothesized    *
*  set of requirements for an Idea Processor (IP), and a     *
*  top-level design to implement those requirements.  An     *
*  Idea Processor is a computer program that helps a         *
*  communicator organize thoughts and ideas in the process   *
*  of composing a document or other form of communication.   *
*  This prototype IP is an implementation of the top-level   *
*  design that was derived from the requirements that were   *
*  originally hypothesized as part of this thesis.  I will   *
*  validate the initial requirements and top-level design    *
*  by observing several people (selected as "experts" in     *
*  communication) as they attempt to use this program to     *
*  complete several writing tasks.  I will record and        *
*  classify any obstacles these people encountered as one    *
*  of four types of "errors": flawed initial requirements    *
*  (Type I), flawed top-level design (Type II), flawed       *
*  implementation of this program (Type III), and problems   *
*  and limitations imposed by the experimental environment   *
*  or the person (Type 0).                                   *
*                                                            *
*************************************************************>
```

```
< ------------------------------------------------------------>
< -----  V A R I A B L E    D E C L A R A T I O N S  ------>
< ------------------------------------------------------------>

    type
        LINK_RECORD = record
                ITEM: string(10) ;
                ROW, COL: integer ;
                NEXT, BACK, SUB: integer ;
                FIRST, SEL: boolean ;
                TEXT: lstring(239) ;
            end ;
        CHARPOINT = ADS of char ;
        ADSWORD = ADS of word ;
        MODES = (TOC,TREE,FACT) ;

    const
        M = 25 ;          < max size of linked list >
        VER = 1.40 ;< current version number >
        ZERO = ADSWORD ( 0, 0 ) ;
        BAR = string (do 38 of chr(218)) ;
        TEE = chr(239)*BAR*chr(221)*BAR*chr(240) ;

    var
< ---------------- Linked List Variables ---------------- >
        LL: array [0..M] of LINK_RECORD ;
        TOP, CURRENT, NEW, FREE: integer ;
        SELECT: array [0..M] of integer ;

< ------------------- Stack Variables -------------------- >
        LEVEL, SP: integer ;
        HISTORY, STACK: array [0..10] of integer ;

< ------------------- Text Variables --------------------- >
        CC, X, Y: integer ;
        FORMAT: integer ;

< ------------------- File Variables --------------------- >
        IDEA_FILE,
        PRINT_FILE: text ;
        FILE_NAME: lstring(30) ;

< ------------------ Other Variables --------------------- >
        COMMAND : integer ;
        COMPOINT : CHARPOINT ;
        TEMP : string(5) ;
        I, J, R, C : integer ;
        INT1, INT2, INT3, INT4 : integer ;
        BSKBD [ external ] : word ;
        MODE : MODES ;
```

```
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >
< : : :   E X T E R N A L   P R O C E D U R E S   : : : : >
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >

procedure BEEP ;  external ;

procedure READKBD(CPOINT : CHARPOINT) ;  external ;

procedure PUTBACKBYTE(BS : adsmem; BY : char ) ; external ;

procedure INITVIDFRAME ( IFR, IC, IL, NC, NL: integer ;
                     BD, BC, BA: byte; FDH, FDW: boolean ) ;
        external ;

procedure INITCHARMAP ( PMAP: ADSWORD; SMAP: word ) ;
        external ;

< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >
< : : : :   M A C H I N E - D E P E N D E N T   : : : : : : >
< : : : : : :     P R O C E D U R E S     : : : : : : : : : >
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >

procedure NORMAL ;
    begin
        write (chr(255),'aa') ;
    end ;

procedure REVERSE ;
    begin
        write (chr(255),'ae') ;
    end ;

procedure BLINK ;
    begin
        write (chr(255),'ai') ;
    end ;

procedure REVERSE_BLINK ;
    begin
        write (chr(255),'am') ;
    end ;

procedure HALF ;
    begin
        write (chr(255),'ab') ;
    end ;

procedure CURSOR(X,Y: integer) ;
    begin
        write (chr(255),'c',chr(X),chr(Y)) ;
    end ;
```

```
procedure ERASE ;
    begin
        write (chr(255),'ef') ;
    end ;

procedure EEOL ;
    begin
        write (chr(255),'el') ;
    end ;

procedure INVISIBLE ;
    begin
        write (chr(255),'vf') ;
    end ;

procedure VISIBLE ;
    begin
        write (chr(255),'vn') ;
    end ;

procedure FRAME ( FNUM : integer ) ;
    begin
        write (chr(255),'x',chr(FNUM) ) ;
    end ;

procedure CURSOR_OFF ;
    begin
        CURSOR( LL[CURRENT].COL, LL[CURRENT].ROW ) ;
        NORMAL ;
        if LL[CURRENT].SEL then BLINK ;
        write ( LL[CURRENT].ITEM ) ;
        NORMAL ;
    end ;

procedure CURSOR_ON ;
    begin
        CURSOR( LL[CURRENT].COL, LL[CURRENT].ROW ) ;
        REVERSE ;
        if LL[CURRENT].SEL then REVERSE_BLINK ;
        write ( LL[CURRENT].ITEM ) ;
        NORMAL ;
        CURSOR(0,0) ;
        EEOL ;
    end ;
```

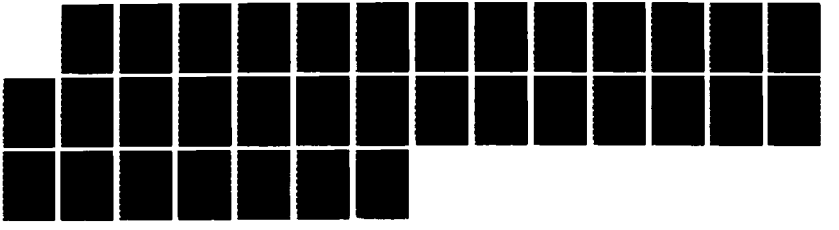
```
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >
< : : :    D I S P L A Y    P R O C E D U R E S    : : : : : : : : : : : >
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >

procedure PUSH(INT: integer) ;
    begin
        STACK[SP] := INT ;
        SP := SP + 1 ;
        if SP > 10 then writeln ('Stack Overflow!!!') ;
    end ;

procedure POP(var INT: integer) ;
    begin
        SP := SP - 1 ;
        if SP < 1 then writeln ('Stack Underflow!!!')
            else INT := STACK[SP] ;
    end ;

procedure DISPLAY_TEXT ;
    begin
        FRAME(1) ; CURSOR(0,0) ; ERASE ;
        write ( LL[CURRENT].TEXT ) ;
        FRAME(0) ;
    end ;
procedure SHOW(I:integer) ;
    begin
        if I=CURRENT then REVERSE ;
        if LL[I].SEL then BLINK ;
        if (I=CURRENT) and (LL[I].SEL) then REVERSE_BLINK ;
        write ( LL[I].ITEM ) ;
        NORMAL ;
    end ;

procedure TRAVERSE(START: integer) ;
    begin
        while START<>0 do
            begin
                LL[START].ROW := R ;
                LL[START].COL := SP*4 ;
                CURSOR ( LL[START].COL, LL[START].ROW ) ;
                SHOW(START) ;
                R := R + 1 ;
                if LL[START].SUB<>0 then
                    begin
                        PUSH(START) ;
                        TRAVERSE(LL[START].SUB) ;
                        POP(START) ;
                    end ;
                START := LL[START].NEXT ;
            end ;
    end ;
```

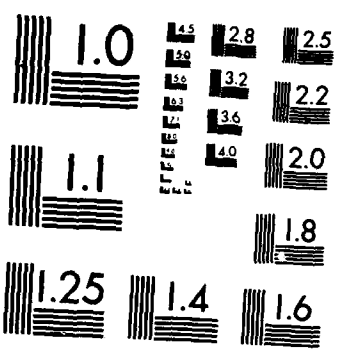
```
procedure ACROSS(S,L: integer) ;
    var
        SS: integer ;
    begin
        SS := S ;
        R := L*2+1 ;
        C := 35 ;
        CURSOR(0,R) ; EEOL ;
        while (not LL[S].FIRST) and
              (LL[S].BACK<>0) and (C>11) do
            begin
                S := LL[S].BACK ;
                C := C-11 ;
            end ;
        if (not LL[S].FIRST) and (LL[S].BACK<>0) then
                write ( chr(27) ) ;
        while (S<>0) and (C<69) do
            begin
                LL[S].ROW := R ;
                LL[S].COL := C ;
                CURSOR(C,R) ;
                SHOW(S) ;
                C := C+11 ;
                S := LL[S].NEXT ;
            end ;
        if S<>0 then
            begin
                CURSOR(79,R) ;
                write ( chr(27) ) ;
            end ;
        CURSOR(0,R+1) ; EEOL ;
        if LL[SS].SUB<>0 then
            begin
                HALF ;
                write ( TEE ) ;
                NORMAL ;
            end else
            begin
                CURSOR(0,R+2) ;
                EEOL ;
            end ;
    end ;

procedure THIS_AND_NEXT (S,L:integer) ;
    begin
        ACROSS(S,L) ;
        if LL[S].SUB<>0 then
            begin
                ACROSS(LL[S].SUB,L+1) ;
            end ;
    end ;
```

```
procedure SHOW_STATUS ;
    begin
        FRAME(0) ;
        CURSOR(0,0) ; write ( 'PIP ',VER:4:2 ) ; EEOL ;
        CURSOR(0,1) ;
        case MODE of
                TOC:  write ( 'Table of Contents' ) ;
                TREE: write ( 'Top-Down Tree' ) ;
            end ;
        EEOL ;
    end ;

procedure REWRITE_DISPLAY ;
    begin
        INVISIBLE ;
        FRAME(0) ;
        SHOW_STATUS ;

        case MODE of

                TOC: begin
                        CURSOR(0,3) ; ERASE ;
                        I := LL[0].NEXT ;
                        R := 3 ;
                        if LL[0].NEXT<>0 then
                            begin
                                TRAVERSE(I) ;
                            end else
                            begin
                                CURSOR(0,3) ;
                                writeln ('empty') ;
                            end ;
                     end ;

                TREE:begin
                        if LL[0].NEXT=0 then
                            begin
                                CURSOR(35,3) ;
                                writeln ('empty') ;
                            end else
                            begin
                               for I := 1 to LEVEL-1 do
                                       ACROSS(HISTORY[I],I) ;
                               THIS_AND_NEXT(CURRENT,LEVEL);
                            end ;
                     end ;

                otherwise write ( 'unknown' ) ;

            end ;  < case >
   end ;
```

AD-A174 421 AN IDEAL DESIGN FOR AN IDEA PROCESSOR(U) AIR FORCE INST 2/2
OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF SYSTEMS AND
LOGISTICS A T MOORE SEP 86 AFIT/GSM/LSH/86S-14

UNCLASSIFIED F/G 9/2 NL





```
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >
< : :  T E X T   E D I T I N G   P R O C E D U R E S  : : >
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >

procedure EDIT ;
    begin
        FRAME(0) ;
        write ( chr(255), 'i3n' ) ;
        FRAME(1) ;
        CURSOR(0,0) ;
        ERASE ;

        write ( LL[CURRENT].TEXT ) ;
        VISIBLE ;
        CURSOR(0,0) ;
        readln ( LL[CURRENT].TEXT ) ;
        INVISIBLE ;

        CURSOR(0,0) ; ERASE ;
        FRAME(0);
        write ( chr(255), 'i3f' ) ;
    end ; < procedure EDIT_SCREEN >

< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >
< : : : :  D O C U M E N T   P R O C E D U R E S  : : : : >
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >

procedure GET_FILENAME ;
    begin
        CURSOR(0,1) ; EEOL ;
        CURSOR(0,0) ; EEOL ; VISIBLE ;
        write ( 'Filename: ' ) ;
        readln ( FILE_NAME ) ;
        INVISIBLE ;
    end ;

procedure SAVE_IDEAS ;
    begin
        GET_FILENAME ;
        CURSOR(0,1) ; write ('SAVING...') ;
        assign ( IDEA_FILE, FILE_NAME ) ;
        rewrite ( IDEA_FILE ) ;
        writeln ( IDEA_FILE, CURRENT :3, ' ',
                        FREE : 3, ' ', LEVEL :2 ) ;
        for I := 0 to 10 do
                write ( IDEA_FILE, HISTORY[I] :3, ' ' ) ;
        writeln ( IDEA_FILE ) ;
```

```
        for I := 0 to M do
            begin
                write   ( IDEA_FILE, LL[I].NEXT :3, ' ' ) ;
                write   ( IDEA_FILE, LL[I].BACK :3, ' ' ) ;
                write   ( IDEA_FILE, LL[I].SUB  :3, ' ' ) ;
                write   ( IDEA_FILE, LL[I].ROW  :2, ' ' ) ;
                write   ( IDEA_FILE, LL[I].COL  :2, ' ' ) ;
                write   ( IDEA_FILE, LL[I].FIRST:5, ' ' ) ;
                writeln ( IDEA_FILE, LL[I].ITEM ) ;
                writeln ( IDEA_FILE, LL[I].TEXT ) ;
            end ;
        close ( IDEA_FILE ) ;
        SHOW_STATUS ;
    end ;

procedure LOAD_IDEAS ;
    begin
        GET_FILENAME ;
        CURSOR(0,1) ; write ('LOADING...') ;
        assign ( IDEA_FILE, FILE_NAME ) ;
        reset ( IDEA_FILE ) ;
        readln ( IDEA_FILE, CURRENT, FREE, LEVEL ) ;
        for I := 0 to 10 do
                read ( IDEA_FILE, HISTORY[I] ) ;
        readln ( IDEA_FILE ) ;
        for I := 0 to M do
            begin
                readln ( IDEA_FILE, LL[I].NEXT, LL[I].BACK,
                         LL[I].SUB, LL[I].ROW, LL[I].COL,
                         LL[I].FIRST, LL[I].ITEM ) ;
                readln ( IDEA_FILE, LL[I].TEXT ) ;
                LL[I].SEL := false ;
            end ;
        close ( IDEA_FILE ) ;
    end ;

procedure OUT_TRAV ( II: integer ) ;
    begin
        while II<>0 do
            begin
                case FORMAT of
                    80,112: begin
                                if LL[II].TEXT.LEN>0 then
                                    write ( PRINT_FILE,
                                        LL[II].TEXT, '  ' );
                            end ;
```

```
                    66,98: begin< bullets >
                               for I := 1 to SP do
                                   write(PRINT_FILE,'  ');
                               for I := 1 to SP do
                                   write(PRINT_FILE,'-');
                               if LL[II].TEXT.LEN>0 then
                                   writeln ( PRINT_FILE,
                                       ' ', LL[II].TEXT )
                                 else
                                   writeln ( PRINT_FILE,
                                       ' ', LL[II].ITEM ) ;
                           end ;
                    otherwise
                           begin
                               writeln ( 'UNKNOWN FORMAT',
                                         ' -- HIT RETURN' );
                               readln ;
                           end ;
                end ;
            if LL[II].SUB<>0 then
                begin
                    PUSH(II) ;
                    OUT_TRAV(LL[II].SUB) ;
                    POP(II) ;
                end ;
            II := LL[II].NEXT ;
        end ;
    end ;

procedure OUT_IDEAS ;
    var
        START: integer ;
    begin
        GET_FILENAME ;
        CURSOR(0,1) ;
        write ( 'FORMAT ( P' ); HALF; write( 'aragraph or');
        NORMAL ; write ( ' B' ); HALF; write ( 'ullets' ) ;
        NORMAL ; write ( ' )' );
        READKBD(COMPOINT) ; FORMAT := ord(COMPOINT@) ;
        CURSOR(0,1) ; EEOL ; write ('PRINTING...') ;
        assign ( PRINT_FILE, FILE_NAME ) ;
        rewrite ( PRINT_FILE ) ;
        START := LL[0].NEXT ;
        OUT_TRAV(START) ;
        close ( PRINT_FILE ) ;
        SHOW_STATUS ;
    end ;
```

```
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >
< : : : : : :   C O R E   P R O C E D U R E S   : : : : : : : >
< : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : >

procedure GET_FREE(var TEMP: integer) ;
    begin
        TEMP := FREE ;
        FREE := LL[FREE].NEXT ;
        if FREE=0 then
            writeln ('GET_FREE is out off free space!') ;
    end ;

procedure GET_NAME ;
    begin
        CURSOR (35,0) ; EEOL ;
        VISIBLE ;
        readln ( LL[CURRENT].ITEM ) ;
        INVISIBLE ;
    end ;

procedure GO_NEXT ;
    begin
        if LL[CURRENT].NEXT<>0 then
            begin
                CURRENT := LL[CURRENT].NEXT ;
                HISTORY[LEVEL] := CURRENT ;
            end ;
    end ;

procedure GO_BACK ;
    begin
        if (not LL[CURRENT].FIRST) and
           (LL[CURRENT].BACK<>0) then
            begin
                CURRENT := LL[CURRENT].BACK ;
                HISTORY[LEVEL] := CURRENT ;
            end ;
    end ;

procedure GO_SUB ;
    begin
        if LL[CURRENT].SUB<>0 then
            begin
                HISTORY[LEVEL] := CURRENT ;
                LEVEL := LEVEL + 1 ;
                if LEVEL > 10 then writeln ('Level > 10') ;
                CURRENT := LL[CURRENT].SUB ;
                HISTORY[LEVEL] := CURRENT ;
            end ;
    end ;
```

```
procedure GO_SUPER ;
    begin
        if LEVEL > 1 then
            begin
                while not LL[CURRENT].FIRST do
                        CURRENT := LL[CURRENT].BACK ;
                if LL[0].NEXT <> CURRENT then
                        CURRENT := LL[CURRENT].BACK ;
                LEVEL := LEVEL - 1 ;
                if CURRENT <> HISTORY[LEVEL] then
                 writeln('Current does not match History!');
            end ;
    end ;

procedure APPEND ;
    begin
        GET_FREE(NEW) ;
        if (LL[CURRENT].NEXT<>0) and
           (LL[CURRENT].BACK<>0) then
                LL[LL[CURRENT].NEXT].BACK:= NEW ;
        LL[NEW].NEXT := LL[CURRENT].NEXT ;
        LL[NEW].BACK := CURRENT ;
        LL[CURRENT].NEXT := NEW ;
        CURRENT := NEW ;
        HISTORY[LEVEL] := CURRENT ;
        GET_NAME ;
    end ;

procedure INSERT ;
    begin
        if CURRENT<>0 then
            begin
                GET_FREE(NEW) ;
                if (LL[CURRENT].BACK<>0) and
                   (LL[CURRENT].FIRST) then
                        LL[LL[CURRENT].BACK].SUB := NEW
                else
                        LL[LL[CURRENT].BACK].NEXT := NEW ;
                LL[NEW].FIRST := LL[CURRENT].FIRST ;
                LL[CURRENT].FIRST := false ;
                LL[NEW].BACK := LL[CURRENT].BACK ;
                LL[NEW].NEXT := CURRENT ;
                LL[CURRENT].BACK := NEW ;
                CURRENT := NEW ;
                HISTORY[LEVEL] := CURRENT ;
                GET_NAME ;
            end ;
    end ;
```

```
procedure MOVE_IDEA (S:integer) ;
    var
        T : integer ;
    begin
        if LL[S].FIRST then
            LL[LL[S].BACK].SUB := LL[S].NEXT
        else
            LL[LL[S].BACK].NEXT := LL[S].NEXT ;
        if LL[S].NEXT<>0 then
            begin
               LL[LL[S].NEXT].BACK := LL[S].BACK ;
               LL[LL[S].NEXT].FIRST := LL[S].FIRST ;
            end ;
        if LL[CURRENT].FIRST then
                LL[LL[CURRENT].BACK].SUB := S
            else
                LL[LL[CURRENT].BACK].NEXT := S ;
        LL[S].FIRST := LL[CURRENT].FIRST ;
        LL[S].NEXT := CURRENT ;
        LL[S].BACK := LL[CURRENT].BACK ;
        LL[CURRENT].BACK := S ;
        LL[CURRENT].FIRST := false ;
    end ;

procedure MOVE_SELECTED ;
    var
        C,
        DE : integer ;
    begin
        C := 0 ;
        while SELECT[C]<>0 do
            begin
                DE := SELECT[C] ;
                if DE<>CURRENT then
                        MOVE_IDEA(DE) ;
                LL[DE].SEL := false ;
                SELECT[C] := 0 ;
                C := C+1 ;
            end ;
       FRAME(0) ;
       write(chr(255),'i8f',chr(255),'i9f',chr(255), 'i0f');
    end ;
```

```
procedure CREATE ;
    begin
        if CURRENT<>0 then
            begin
                HISTORY[LEVEL] := CURRENT ;
                LEVEL := LEVEL + 1 ;
                if LEVEL > 10 then writeln ('Level > 10') ;
                GET_FREE(NEW) ;
                LL[NEW].BACK := CURRENT ;
                LL[CURRENT].SUB := NEW ;
                LL[NEW].FIRST := true ;
                LL[NEW].NEXT := 0 ;
                CURRENT := NEW ;
                HISTORY[LEVEL] := CURRENT ;
                GET_NAME ;
            end ;
    end ;

procedure SUBORDINATE_IDEA (S:integer) ;
    var
        T : integer ;
    begin
        if LL[S].FIRST then
            LL[LL[S].BACK].SUB := LL[S].NEXT
        else
            LL[LL[S].BACK].NEXT := LL[S].NEXT ;
        if LL[S].NEXT<>0 then
            begin
               LL[LL[S].NEXT].BACK := LL[S].BACK ;
               LL[LL[S].NEXT].FIRST := LL[S].FIRST ;
            end ;
        if LL[CURRENT].SUB=0 then
            begin
                LL[S].FIRST := true ;
                LL[S].BACK := CURRENT ;
                LL[S].NEXT := 0 ;
                LL[CURRENT].SUB := S ;
            end else
            begin
                T := LL[CURRENT].SUB ;
                while LL[T].NEXT<>0 do
                        T := LL[T].NEXT ;
                LL[S].FIRST := false ;
                LL[S].BACK := T ;
                LL[S].NEXT := 0 ;
                LL[T].NEXT := S ;
            end ;
    end ;
```

```
procedure SUBORDINATE_SELECTED ;
    var
        C,
        DE : integer ;
    begin
        C := 0 ;
        while SELECT[C]<>0 do
            begin
                DE := SELECT[C] ;
                if DE<>CURRENT then
                        SUBORDINATE_IDEA(DE) ;
                LL[DE].SEL := false ;
                SELECT[C] := 0 ;
                C := C+1 ;
            end ;
        FRAME(0) ;
        write(chr(255),'i8f',chr(255),'i9f',chr(255),'i0f');
    end ;

procedure ZERO_ITEM (II:integer) ;
    begin
        LL[II].BACK := 0 ;
        LL[II].SUB := 0 ;
        LL[II].FIRST := false ;
        LL[II].SEL := false ;
        LL[II].TEXT := NULL ;
    end ;

procedure DELETE_IDEA(D : integer) ;
    begin
        if LL[D].SUB=0 then
            begin
                if LL[D].FIRST then
                    LL[LL[D].BACK].SUB :=
                                LL[D].NEXT
                else
                    LL[LL[D].BACK].NEXT :=
                                LL[D].NEXT ;
                if LL[D].NEXT<>0 then
                    begin
                       LL[LL[D].NEXT].BACK :=
                                LL[D].BACK ;
                       LL[LL[D].NEXT].FIRST :=
                                LL[D].FIRST ;
                    end ;
                NEW := D ;
                if D=CURRENT then
                    begin
                        if LL[CURRENT].NEXT<>0 then
```

```
                                begin
                                    CURRENT := LL[CURRENT].NEXT;
                                end else
                                begin
                                    if LL[CURRENT].FIRST then
                                        begin
                                            GO_SUPER ;
                                        end else
                                        begin
                                            CURRENT :=
                                               LL[CURRENT].BACK;
                                        end ;
                                end ;
                            HISTORY[LEVEL] := CURRENT ;
                            CURSOR(0,20) ;
                            write ('CURRENT=',CURRENT) ;
                        end ;
                    ZERO_ITEM(NEW) ;
                    LL[NEW].NEXT := FREE ;
                    FREE := NEW ;
                end ;
        end ;

    procedure DELETE_SELECTED ;
        var
            C,
            DE : integer ;
        begin
            C := 0 ;
            while SELECT[C]<>0 do
                begin
                    DE := SELECT[C] ;
                    if DE<>CURRENT then
                            DELETE_IDEA(DE) ;
                    LL[DE].SEL := false ;
                    SELECT[C] := 0 ;
                    C := C+1 ;
                end ;
            FRAME(0) ;
            write(chr(255),'i8f',chr(255),'i9f',chr(255),'i0f');
        end ;
```

```
procedure TOGGLE_SELECT ;
    var
        SEARCH : integer ;
        FOUND : boolean ;
    begin
        SEARCH := 0 ;
        FOUND := false ;
        while (SELECT[SEARCH]<>0) and (not FOUND) do
            begin
                if SELECT[SEARCH]=CURRENT then
                    begin
                        FOUND := true ;
                        while SELECT[SEARCH]<>0 do
                            begin
                                SELECT[SEARCH] :=
                                        SELECT[SEARCH+1] ;
                                SEARCH := SEARCH+1 ;
                            end ;
                        LL[CURRENT].SEL := false ;
                    end else
                    begin
                        SEARCH := SEARCH+1 ;
                    end ;
            end ;
        if not FOUND then
            begin
                SELECT[SEARCH] := CURRENT ;
                SELECT[SEARCH+1] := 0 ;
                LL[CURRENT].SEL := true ;
            end ;
        if SELECT[0]<>0 then
            begin
                write ( chr(255), 'i8n', chr(255),
                            'i9n', chr(255), 'i0n' ) ;
            end else
            begin
                write ( chr(255), 'i8f', chr(255),
                            'i9f', chr(255), 'i0f' ) ;
            end ;
    end ;
```

```
procedure INITIALIZE ;
    var
        J : integer ;
    begin

        INITVIDFRAME(0,0,4,80,24,0,0,0,false,false) ;
        INITVIDFRAME(1,0,0,80, 3,4,4,1,false,false) ;
        INITCHARMAP(ZERO,4640) ;

        FRAME(1) ; CURSOR(0,0) ; ERASE ;
        FRAME(0) ; CURSOR(0,0) ; ERASE ;
        write ( chr(255), 'pf' ) ;

        for J := 0 to M do
            begin
                LL[J].NEXT := J+1 ;
                ZERO_ITEM(J) ;
            end ;

        LL[0].NEXT := 0 ;
        CURRENT := 0 ;
        LEVEL := 1 ;
        SP := 1 ;
        FREE := 1 ;
        MODE := TREE ;
        SELECT[0] := 0 ;
        REWRITE_DISPLAY ;
    end ;

procedure TOGGLE_MODE ;
    begin
        case MODE of

                TOC : begin
                        MODE := TREE ;
                        CURSOR(0,0) ; ERASE ;
                      end ;

                TREE: MODE := TOC ;

                otherwise writeln ('Lost Mode Mode') ;

            end ;
    end ;
```

```
< ========================================================== >
< ====================== MAIN PROGRAM ====================== >
< ========================================================== >

    begin

        INITIALIZE ;

        while COMMAND <> 4 do

            begin

                CURSOR(0,0) ; NORMAL ;

                READKBD(COMPOINT) ;
                COMMAND := ord(COMPOINT^) ;

                case COMMAND of

                        11 : case MODE of< down arrow >
                                TOC : begin
                                        CURSOR_OFF ;
                                        GO_NEXT ;
                                        CURSOR_ON ;
                                        DISPLAY_TEXT ;
                                      end ;
                               TREE: begin
                                        GO_SUB ;
                                        REWRITE_DISPLAY ;
                                        DISPLAY_TEXT ;
                                      end ;
                             end ;

                         1 : case MODE of< up arrow >
                                TOC : begin
                                        CURSOR_OFF ;
                                        GO_BACK ;
                                        CURSOR_ON ;
                                        DISPLAY_TEXT ;
                                      end ;
                               TREE: begin
                                        GO_SUPER ;
                                        REWRITE_DISPLAY ;
                                        DISPLAY_TEXT ;
                                      end ;
                             end ;
```

```
18 : case MODE of< right arrow >
        TOC : begin
                CURSOR_OFF ;
                GO_SUB ;
                CURSOR_ON ;
                DISPLAY_TEXT ;
              end ;
        TREE: begin
                GO_NEXT ;
                THIS_AND_NEXT
                   (CURRENT,LEVEL) ;
                DISPLAY_TEXT ;
              end ;
      end ;

14 : case MODE of< left arrow >
        TOC : begin
                CURSOR_OFF ;
                GO_SUPER ;
                CURSOR_ON ;
                DISPLAY_TEXT ;
              end ;
        TREE: begin
                GO_BACK ;
                THIS_AND_NEXT
                    (CURRENT,LEVEL);
                DISPLAY_TEXT ;
              end ;
      end ;

30 : begin       < F9 >
        if SELECT[0]=0 then
                INSERT
            else
                MOVE_SELECTED ;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
      end ;

29,127 : begin< F8 or delete >
        if SELECT[0]=0 then
                DELETE_IDEA(CURRENT)
            else
                DELETE_SELECTED ;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
      end ;
```

```
31 : begin       < F10 >
        if SELECT[0]=0 then
              CREATE
           else
              SUBORDINATE_SELECTED;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
     end ;

 4 : begin       < finish >
        FRAME(1) ; CURSOR(0,0);
        ERASE ; CURSOR(35,1) ;
        writeln ('exiting...') ;
     end ;

32..126 :       < alphanumeric keys >
    begin
        PUTBACKBYTE ( ads BSKBD,
                      chr(COMMAND) );
        APPEND ;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
    end ;

23 : begin       < F3 >
        EDIT ;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
     end ;

21 : begin       < F1 >
        TOGGLE_MODE ;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
     end ;

 2 : begin        < mark >
        TOGGLE_SELECT ;
        CURSOR(LL[CURRENT].COL,
               LL[CURRENT].ROW) ;
        SHOW(CURRENT) ;
     end ;

28 : begin       < F7 >
        SAVE_IDEAS ;
     end ;
26 : begin       < F6 >
        LOAD_IDEAS ;
        REWRITE_DISPLAY ;
        DISPLAY_TEXT ;
     end ;
```

```
                    25 : begin      < F5 >
                            INITIALIZE ;
                         end ;

                    24 : begin      < F4 >
                            OUT_IDEAS ;
                         end ;

                    otherwise
                        begin
                            BEEP ;
                        end ;

                end ; < case >

        end ; < while >

end. < program >
```

APPENDIX B

VALIDATION TASKS

PRACTICE TASK (18):

Seeds are usually planted in field nurseries, and the resulting seedlings are transplanted to the production field.

Production is extensive in Western Europe, where brussels sprouts are a popular vegetable.

Best growth occurs during an extended period of cool temperatures, and 90-120 days are required from transplanting to harvest.

Brussels sprouts, Brassica oleracea, vegetables belonging to the mustard family, Cruciferae, are closely related to cabbage, cauliflower, collards, broccoli, and kohlrabi.

The plant is biennial, producing an upright stout stem bearing many long petioled, round, waxy leaves and vegetative buds (the sprouts) in the leaf axils in the first year.

They are so named because they were first grown in quantity near Brussels, Belgium, during the 16th century.

Conspicuous yellow flowers are produced in the second year.

Each plant requires about 60-90 cm (24-36 in) of field space.

Wild cabbage is the common ancestor for these plants, which are collectively termed cole crops.

U.S. commercial production is limited to the midcoastal area of California and some areas of New York State.

Brussels sprouts are grown for sale both fresh and frozen.

Multiple hand-harvests have now been largely replaced by once-over harvests in which mechanical aids are used to remove the sprouts from the stems.

FIRST TASK:

Jupiter is the largest planet, a massive gas giant noted for its Great Red Spot.

As the planet closest to the sun, Mercury is a scorched, barren world.

A planet is a nonluminous body that revolves around a central star.

Normally, this planet is the furthest from the sun, an icy-cold rock. However, it is currently within the orbit of Neptune due to the eccentricity of its orbit.

The inner planets, Mercury, Venus, Earth, and Mars, are generally small and solid bodies.

A very odd planet, Uranus has its poles tilted almost directly toward the Sun and a faint ring system.

Venus is blanketed by dense clouds; it is largely through radar that we know of its surface features.

Though usually depicted in science fiction literature as being inhabited by 'little green men,' Mars is an unlikely place for life.

Gas giants characterize the outer planets of the solar system: Jupiter, Saturn, Uranus, Neptune, and Pluto.

Neptune is Voyager's next stop on its interplanetary journey.

At just the right distance from the Sun, the Earth enjoys a suitable environment for the development of life.

The rings of Saturn have been investigated and shown to be quite complex.

SECOND TASK (5) :

During the third stroke, the compressed mixture is ignited--either by compression ignition or by spark ignition. The heat produced by the combustion causes the gases to expand within the cylinder, thus forcing the piston downward.

In the four-stroke cycle, also known as the Otto cycle, the downward movement of a piston located within a cylinder creates a partial vacuum.

The piston's connecting rod transmits the power from the piston to the crankshaft. This assembly changes reciprocating--in other words, up-and-down or back-and-forth motion--to rotary motion.

On the first stroke, the intake valve is opened while the exhause valve is closed; atmospheric pressure forces a mixture of gas and air to fill the chamber.

The operation of the internal-combustion reciprocating engine employs either a four-stroke or a two-stroke cycle. (A stroke is one continuous movement of the piston within the cylinder.)

On the fourth stroke, the exhaust valve is opened so that the burned gases can escape as the piston moves upward; this prepares the cylinder for another cycle.

On the second stroke, the intake and exhaust valves are both closed as the piston starts upward. The mixture is compressed from normal atmospheric pressure to between 4.9 and 8.8 kg/sq cm.

The four strokes are called, in order of sequence, intake, compression, power, and exhaust.

Two valves located inside the combustion chamber are controlled by the motion of a camshaft connected to the crankshaft.

Internal-combustion spark-ignition engines having a two-stroke cycle combine intake and compression in a single first stroke and power and exhaust in a second stroke.

APPENDIX C

VALIDATION OBSERVATIONS AND NOTES

Participant #1

- need UNDO function
- would prefer to OPEN and CLOSE files like WRITEone
- DELETE-selected doesn't delete the current (highlighted) idea
- Status information disappears in TOC mode
- expected to use GO instead of RETURN, like Burroughs software
- prefers TOC mode (can see all ideas)
- moved "top" idea to first in list
- SUBORDINATE-selected appends to sub-ideas
- need APPEND-selected function
- unlimited bullet length
- didn't like TREE mode: like need to search directories in UNIX
- sub-topics not centered under topics in TREE mode
- need CUT and PASTE in EDIT mode
- SEARCH function
- wants top few lines of text at top of screen, with scrolling
- relational data base functions
- index card features
- paragraph-oriented: text-editing of screen at a time
- showed intuitive grasp of IP, appreciated and used capabilities

Participant #2

- "CURRENT=" bug at bottom of screen
- wanted EDIT function
- need to be able to abort SUBORDINATE and other idea-entry
- need UNSUBORDINATE function
- worked hierarchically
- SUBORDINATE wipes out existing sub-topics
- wants to select and move text between ideas
- In TREE mode, current level plus two or more is NOT erased, and is often inconsistant with previous levels
- wants to move directly through ideas in TOC mode (or even TREE)
- at one point, had up to six levels of subordination
- CUT and PASTE ideas
- would like to see entire tree in TREE mode
- wants to review whole paragraph
- used selected-order feature
- likes idea of auto-bibliography reference
- wants a storehouse of 100-200 sentences (in columns on screen), from which he could select the ones for a paragraph, and then go into TOC or TREE mode to manipulate ideas
- wants deleted items to go into an "unused" category

Participant #3

- preferred TOC mode (to see all ideas)
- liked to add new ideas and associated text
- moved "top" idea to beginning of list

- used ordered-select
- uses index cards
- likes text size of 3 lines
- would prefer integrated IP and word processing
- wants UNDO function

Participant #4

- status line erased in TOC mode
- mostly "looking around" during practice
- a little confused about TOC mode cursor controls
- third level ideas incorrect/not erased in TREE mode
- flipped between modes often to help learn TOC mode
- SUPERORDINATE command
- DELETE-ALL to delete topic and all sub-topics under it
- IP crashed when trying to subordinate an idea under its own sub-topic
- SUBORDINATE removes existing sub-topics
- wants to review in TOC mode sequentially
- SWITCH function to swap two ideas
- HELP function
- moved "top" idea to the first in list
- preferred to move ideas one-at-a-time (didn't use INSERT-selected)
- put ideas in order before subordinating them
- did not like TREE mode (wants to see all topics)
- need APPEND-selected
- UNSUBORDINATE

- up to four levels of subordination at one point

- difficult to handle transitional sentences or thoughs in a hierarchical structure for a paragraph

- wants to be able to read text for a particular idea without having to move the cursor back

- sometimes relationship between ideas is not superior/subordinate/equal, but factorially different

- does classification task distract user from ultimate purpose

- needs a REVIEW-selected (partial or temporary merging of several ideas to see if they would flow)

- does drafts on paper first

- would prefer a few more text lines

- on-screen numbering of ideas

Participant #5

- confused by having to hit SUBORDINATE or INSERT before entering idea name; would prefer typing the idea, then hitting SUBORDINATE, INSERT, or APPEND

- confused by movement of ideas under highlighted cursor in TREE mode

- wanted APPEND key

- thought that an idea had to maintain its "level" (couldn't unsubordinate by INSERT-selected)

- SUBORDINATE destroys existing sub-topics (should APPEND)

- MOVE key (INSERT-selected not intuitive)

- things jump around too much in TREE mode

- doesn't use advanced features of programs, just basic ones

- puts great importance on "key words" (idea names)

- third-level ideas not consistant/erased

- "CURRENT=" bug
- moved "top" idea to first in list
- used ordered-select feature
- didn't like having to go back to one idea to re-read its text
- used TREE mode, even to review flow of text
- wanted to group similar items, because too many on main line (information hiding)
- wants RENAME function (to edit idea names)
- subordinated ideas one-at-a-time
- preferred to re-write sentences, rather than trying to reorder existing ones
- began to use IP spontaneously to organize topics for a paper he was working on
- liked to outline top-down (level by level)
- would prefer to edit entire paragraphs, and then move them
- still prefers pen-and-paper for outlining and drafting
- saw a use for IP in specification writing
- used TREE mode primarily because I defaulted to it

Participant #6

- couldn't APPEND-selected
- can't DELETE-selected item under cursor (highlighted)
- moved "top" idea to first in list
- ordered entire list, but NEVER SUBORDINATED any ideas
- need to abort out of INSERT, SUBORDINATE, APPEND idea functions
- used sub-topics as "notes" to herself on order of ideas

- moved ideas one-at-a-time
- need TOP and BOTTOM commands
- appears to think in a linear-sequential way
- wanted REVIEW capability
- wanted five lines of text
- wanted to be able to step through ideas sequentially
- likes automatic bibliography/reference idea
- used TOC mode

Participant #7

- third-level ideas inconsistant/not erased
- "CURRENT=" bug
- can't DELETE-selected current (highlighted) idea
- liked to add new ideas
- need to be able to abort INSERT/APPEND/SUBORDINATE idea entry, and filter non-printing characters out
- need APPEND-selected
- need UNMARK (to unselect one or all ideas)
- liked to unsubordinate by moving
- wants to review ideas sequentially in TOC mode
- UNDO command
- doesn't like ideas scrolling under cursor in TREE mode
- doesn't like limit of 25 ideas
- likes TREE mode for information hiding, especially in a many-levelled outline
- need to be able to delete-to-end, or MARK-BOUND, so that you don't have to select each idea individually

- preferred TOC mode overall

- thought APPEND-selected should be RETURN

- prefers a few good commands over many complex ones

- subordinated ideas first, then put them in order

- liked blinking ideas when marked

- didn't like going back to read text of another idea; "lousey short-term memory..."

- wanted to be able to select and move text to idea names, especially when lists are embedded in the text. Didn't see a need to go the other way (idea names to text)

- wanted Idea Processing and Word Processing accessable from each other, but distinct

- wants 5 or 6 lines of text

Participant #8

- at one point, subtopics became linked with the top idea, where you could move the cursor from the top idea "back" to its previous neighbor but end up back down in the sub-topics of an idea near the end of the list

- need to be able to abort entering a new idea (INSERT/SUBORDINATE)

- "CURRENT=" bug

- wants COPY-selected function

- third-level ideas inconsistant/not erased in TREE mode

- moved "top" idea to first in list

- built hierarchy top-down, using ordered-select feature

- likes to write sentences, then create paragraphs, then move paragraphs; appears to think, research, and gather data in a bottom-up approach, but then write/translate in a top-down manner

- does paragraphs first (before organizing)

- prefers TOC mode

- wants full page (screen) editor, or at least half of a screen

- likes to do ordering last, at the highest level (bottom-up: text/content, then structure)

- would not use IP without a word-processing function (in fact, thought of IP as a part of the WP)

- said that when he types his text in, it is pretty close to final form

- does NOT outline; knows general direction, but lets research re-define its own topic (process of discovery)

- ordered ideas first, then subordinated them top-down

- would use bullet format with whole paragraphs (ala Fenno), but likes to have options

- would use IP on one chapter at a time, not whole thesis

Bibliography

1. Bee, Helen L., and Sandra K. Mitchell. The Developing Person: A Life-Span Approach. San Francisco: Harper & Row, Publishers, Inc., 1980.

2. Bereiter, Carl. "Development In Writing," Cognitive Processes In Writing, edited by L. W. Gregg and E. R. Steinberg. New Jersey: Lawrence Erlbaum Associates, Inc., 1980.

3. Burroughs B-20 Systems Operating System (BTOS) Reference Manual, Volume 1 (Rel. 4.0). Document No. 1171675. Burroughs Corporation, Detroit MI, 1984.

4. Collins, Allan, and Dedre Gentner. "A Framework for a Cognitive Theory of Writing," Cognitive Processes In Writing, edited by L. W. Gregg and E. R. Steinberg. New Jersey: Lawrence Erlbaum Associates, Inc., 1980.

5. Duffy, Joseph W. "Internal Combustion Engine," Academic American Encyclopedia, Grolier Electronic Publishing, Inc., 1986.

6. Fenno, Charles R. "The Minor Aches and Pains of the Electronic Office: Professional Writers and Editors Talk About Their Work," Proceedings of the 33rd International Technical Communication Conference. 352-356. Washington, Society for Technical Communication, 1986.

7. Flower, Linda S., and John R. Hayes. "The Dynamics of Composing: Making Plans and Juggling Constraints," Cognitive Processes In Writing, edited by L. W. Gregg and E. R. Steinberg. New Jersey: Lawrence Erlbaum Associates, Inc., 1980.

8. Friendly, Michael L. "In Search of the M-Gram: The Structure of Organization in Free Recall," Cognitive Psychology, 9: 188-249 (1977).

9. Good, Michael. "Etude and the Folklore of User Interface Design," Proceedings of the ACM SIGPLAN/SIGOA Symposium on Text Manipulation. 34-43. Order number 548810. Oregon, Special Interest Group on Office Automation, Association of Computing Machinery, June 1981.

10. Gorman, Ronald H., and H. Kent Baker. "Brainstorming Your Way To Problem-Solving Ideas," Personnel Journal: 438-440 (August 1978).

11. Gould, John D. "Experiments on Composing Letters: Some Facts, Some Myths, and Some Observations," Cognitive Processes In Writing, edited by L. W. Gregg and E. R. Steinberg. New Jersey: Lawrence Erlbaum Associates, Inc., 1980.

12. Grossman, Stephen R. "Brainstorming Updated." Training and Development Journal, 38 (2): 84-87 (February 1984).

13. Grunig, James E., et al. "An Axiomatic Theory of Cognition and Writing," Journal of Technical Writing and Communication, 15 (2): 95-130 (1985).

14. Harrington, Henry R., and Richard E. Walton. "The Warnier-Orr Diagram for Designing Essays," Journal of Technical Writing and Communication, 14: 193-201 (1984).

15. Hayes, John R., and Linda S. Flower. "Identifying the Organization of Writing Processes," Cognitive Processes In Writing, edited by L. W. Gregg and E. R. Steinberg. New Jersey: Lawrence Erlbaum Associates, Inc., 1980.

16. Herrstrom, David S. "Technical Writing as Mapping Description onto Diagram: The Graphic Paradigms of Explanation.," Journal of Technical Writing and Communication, 14 (3): 223-240 (1984).

17. Huber, George P. "Cognitive Style as a Basis for MIS and DSS Designs: Much Ado About Nothing?" Management Science, 29 (5): 567-579 (May 1983).

18. Maynard, Donald N. "Brussels Sprouts," Academic American Encyclopedia, Grolier Electronic Publishing, Inc., 1986.

19. Minor, Dennis E. "Coherent Deformation in Scientific and Technical Writing," Journal of Technical Writing and Communication, 10 (3): 201-212 (August 1980).

20. Peels, Arno J. H. M., and others. "Document Architecture and Text Formatting," ACM Transactions on Office Information Systems, 3 (4): 347-369 (1985).

21. Piaget, J. "Development and Learning," Piaget Rediscovered, edited by R. Ripple and V. Rockcastle. Ithaca: Cornell University Press, 1964.

22. Reitman, Walter R. Cognition and Thought: An Information Processing Approach. New York: John Wiley & Sons, Inc., 1965.

Vita

Captain Andrew T. Moore was born on 31 December 1960 in Nashville, Tennessee. After graduating from high school in Shreveport, Louisiana, he attended the United States Air Force Academy in Colorado Springs, Colorado. He graduated in June 1982 with a Bachelor of Science in Computer Science. For the next two years he worked in the Deputy for Engineering, Aeronautical Systems Division (ASD/ENASF) validating MIL-STD-1750A standard embedded computers, after which he was collocated to the Deputy for Reconnaisance and Electronic Warfare (ASD/RWEA) to work on the PLSS and IRST programs. In May 1985 he entered the School of Systems and Logistics, Air Force Institute of Technology, under the Graduate Systems Management program.

Temporary Address: 1ISG/XR
Pentagon ADM, VA 20330

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

ADA174421

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GSM/LSH/86S-14	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Systems and Logistics; 6b. OFFICE SYMBOL (If applicable): AFIT/LSA	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Andrew T. Moore, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1986 September; 15. PAGE COUNT: 127	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Idea Processor, Thought Processor, Composition, Technical Communication, Text Processing
05	08		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: AN IDEAL DESIGN FOR AN IDEA PROCESSOR

Advisor: Charles R. Fenno
Associate Professor of Research and Communication

Approved for public release: IAW AFR 190-1
LYNN E. WOLAVER 29 Sept 86
Dean for Research and Professional Development
Air Force Institute of Technology (AFSC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Charles R. Fenno	22b. TELEPHONE NUMBER (Include Area Code): 513-255-6761; 22c. OFFICE SYMBOL: AFIT/LSH

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

This research determined requirements for an idea processor, a computer program that supports the process of composition by helping the user generate, organize, translate, and review ideas as they are prepared for written or verbal communication. These requirements were derived from theories and models of composition found in a review of current literature. An initial design for an idea processor was developed to meet these requirements, based on features of existing software packages and on the hardware capabilities of microcomputers in use in the Air Force, such as the Zenith Z-100. To implement this initial design, a prototype idea processor was developed on the Burroughs B-26 microcomputer system at AFIT.

The original requirements and design were validated in an exploratory experiment. Eight graduate students--identified by communication instructors as "expert" communicators--performed example idea processing tasks using the prototype idea processor. These participants verbalized any perceived "obstacles" to their own preferred style of composition. The participants' comments were recorded and classified according to four types of obstacles: missing requirements (type 1), incomplete design (type 2), faulty implementation (type 3), or the limitations of each person's writing ability or the testing environment (type 0).

The original idea processor design was modified to remove the type 2 obstacles found in the validation. Solutions to other obstacles and general comments by the participants were presented as suggested improvements to the prototype idea processor, and directions for further research were discussed.

END

1-87

DTIC