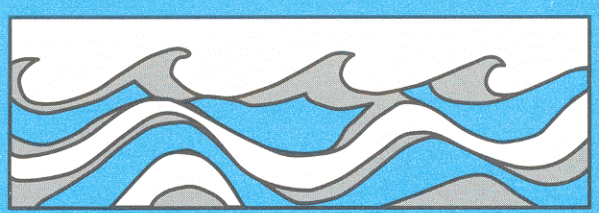
University of Washington
Department of Civil and Environmental Engineering

INLET: AN INTERACTIVE NATURAL LANGUAGE ENVIRONMENT FOR A WATER RESOURCES DATABASE

Lynn R. Spence
Richard N. Palmer



Water Resources Series
Technical Report No.112
August 1989

Seattle, Washington
98195

Department of Civil Engineering
University of Washington
Seattle, Washington 98195

INLET: AN INTERACTIVE NATURAL LANGUAGE ENVIRONMENT FOR A WATER RESOURCES DATABASE

Lynn R. Spence
Richard N. Palmer

Water Resources Series
Technical Report No. 112

August 1989

INLET:
AN INTERACTIVE NATURAL LANGUAGE ENVIRONMENT FOR A WATER RESOURCES DATABASE

by

Lynn R. Spence and Richard N. Palmer

Department of Civil Engineering
University of Washington, FX-10
Seattle, Washington 98195

Project Completion Report Submitted to:

The State of Washington Water Research Center and
The U.S. Department of the Interior

Water Research Center Project No. A-159-WASH
U.S. Department of the Interior Grant No. G-1597

For the Period
May 1, 1989 through March 31, 1989

August, 1989

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The work upon which this publication is based was supported in part by the Department of the Interior, U.S. Geological Survey, through the State of Washington Water Research Center.

The study was supervised by Richard N. Palmer, Associate Professor of Civil Engineering and performed by Lynn R. Spence, Graduate Research Assistant in the Department of Civil Engineering. The study served as the basis for Ms. Spence's MSE thesis.

The authors acknowledge the continued support of the Seattle Water Department and in particular the suggestions of Julie Cohan. The principal investigator would also like to thank Dr. Ramon Lopez de Mantaras and the Centro de Estudios Avanzados in Blanes, Spain, for providing him an office and the opportunity to initiate work on this paper while on sabbatical leave from the University of Washington.

The contents of this publication do not necessarily reflect the views and policies of the Department of the Interior, nor does mention of trade names or commercial products constitute their endorsement by the United States Government.

CHAPTER 1
INTRODUCTION AND RESEARCH OBJECTIVES

INTRODUCTION

This report describes the development of a water resource planning effort that evolved rapidly over time. The topic is the development of water resource management models for use in drought planning for the Seattle, Washington, water supply. The effort began as a traditional study to determine the yield of the reservoirs that serve as a primary source of water for Seattle and the proper operating policies for these reservoirs during period of drought. However, it was soon discovered that this approach did not capture either the confidence nor imagination of the water supply managers. Because of this, new and emerging computer technology was applied to make the results of the analytical models more meaningful to those responsible for the results of the decisions being made. To do this, the new models had to make information directly available to the managers, present it in a way that they could understand, and allow them to see the effects of their decisions. What is described in the report is the process in which the tools used to solve a problem were adjusted to more closely meet the needs of the users, rather than the reverse. In doing so the tools became a vital part of the framework used to make decisions.

The water resource planning profession, like its counterparts in other fields of planning, is undergoing a major revolution in the way in which decisions are made. This revolution represents the second wave of a fundamental change that occurred with the development of the digital computer, a device that drastically altered the way in which information used for decision-making can be collected,

analyzed, manipulated, and reported. With the initial availability of the computer, water resource planners were able to develop quantitative plans, simulate projected futures, and formulate models that explored "optimal" solutions to complex problems. These computational tasks were virtually impossible on the same scale a mere decade before.

A price was paid, however, for these new capabilities. The models used for decision-making soon became the central figure in the planning process. An elite group of engineers and technicians, whose job it was to develop, maintain, and nurture these models soon evolved, and although expert at their job, this group often did not have special training in water resources management or decision-making. The models that these individuals developed often appeared to be rather inflexible in character, limited in scope, and in constant need of their developer. Because of these limitations, the models were often ignored.

The second wave of computer applications, the topic of this paper, concerns the development of models whose purpose it is to eliminate these basic difficulties and to return the decision-maker to the center of the planning process. More fundamentally, these models attempt to blend the insights and skills of the decision-maker and the computational dexterity and logic of the computer into an interactive blend in which the proper questions can be posed by the decision-maker and answered quickly and accurately by the computer. Because the interaction between the decision-maker and the computer is rapid and flexible, the decision-maker is encouraged to explore new possibilities that result from the previous answers supplied by the computer.

It should be noted that this second wave of computer usage has been created by not only a perception that previous models have failed in situations in which they

could have been vastly more useful, but by yet another advance in computer technology. During the past decade, there as been a truly unprecedented change in both the computer hardware and software available for water resource planning. Today, desk-top engineering workstations are available for less that $20,000 that are more powerful than mainframe computer of only a decade ago. As these workstations become even more powerful and their costs continue to decrease, the proliferation of these engineering workstations will increase the number of computer-literate decision-makers and change the way in which these decision-makers use computers. No longer willing to accept the idea that computer programs are black boxes beyond their understanding, decision-makers will demand tools that reflect more closely their perceptions of the problem and address it in a way that they believe to be realistic and meaningful.

The types of models described in this report have also evolved during the last decade. Originally such models were denoted as *interactive models* to distinguish them from more traditional batch models in which the user does not interact with the model in any way while it was being executed. These models also have come to include expert system models, which are fundamentally different than procedural, interactive models. Further, many elements of artificial intelligence have begun to be applied to the field of water resources management.

This report describes the development of INLET, an Interactive Natural Language Environment applied to a water resources database that has been created for the Seattle water supply. INLET is a database interface that accepts commands and questions posed in ordinary (conversational) English, and a menu-driven query system. By using ordinary English commands, water resource managers and staff who are not familiar with either using a formal database query system or computer

programming can easily access hydrologic and management data and use the analytical, statistical, and graphical capabilities provided by INLET. This interface assumes the user is familiar with the problem domain but it does not assume the user has computer expertise in any particular area.

RESEARCH OBJECTIVE

The primary objective of this research is to explore the use of natural language interfaces for water resources planning and management computer models and data analysis tools. Obtaining this objective will improve the planning process by more effectively incorporating both quantitative and qualitative information into all phases of water resource management. The result of this research is the development of highly user-friendly software that increases the access, availability, and use of water resource planning data to managers.

This objective has been achieved through the development of INLET, an Interactive Natural Language EnvironmenT, that allows convenient, rapid, and extensive access to water resource management data. This system is designed to require a minimum of training to use, thus making it immediately available for water managers who do not have the time or interest to be trained to access traditional databases. It must be recognized that water resource planning and management requires analysis of large amounts of both quantitative and subjective information. A significant amount of this data is quantitative, such as streamflow records and system characteristics. An important portion of information, however, is subjective; that is, it can not be adequately described in quantitative terms. Such information includes the quality of streamflow forecasts, the uncertainty associated with the public's response to a call for water-use restrictions, or the degree to which a current drought is similar to events in the past.

Access to all relevant information, both quantitative and qualitative, is required for informed decision-making. Analysis of this data is also required. Some commonly performed analyses include the calculation of streamflow statistics, the plotting of streamflows, and comparisons of alternative operating policies. When performed by traditional techniques, such analysis requires extensive training and experience and the understanding of a formal query procedure of the database. Infrequent use of the database by managers may discourage them from investing the time and effort to learn it, thus prohibiting their direct access to the information and decreasing the probability that the information will actually be used in making a final decision. It is the objective of this research, in the development of INLET, to increase access to such databases and improve the decision-making process.

The remainder of this report is organized as follows: Chapter 2 provides a literature review of relevant topics that includes interactive computing in water resource planning and a brief history of natural languages and their use in database access. Chapter 3 provides an overview of the Seattle water supply including system configuration, operational considerations, and existing management procedures. The chapter also includes a summary of previous studies to date. Chapter Four introduces the computer code, INLET, and describes its development. Chapter Five illustrates the use of this software and provides evidence of its value in the planning process. Chapter Six presents the conclusions of this research and the suggested areas of new research.

CHAPTER 2

LITERATURE REVIEW

Water resource planners and decision-makers must be directly involved in the development of any computer model designed for their use. Their involvement increases the probability that the computer model addresses questions of importance, is developed so that it can be successfully used, and includes all relevant data. In order to include water resource planners in model development and encourage the model's eventual use, highly user-friendly computer software is required. This chapter briefly reviews traditional approaches in water resources modeling and how those approaches have limited the success of many models as applied planning and operational tools. Natural language interfaces, a recent technology from the field of artificial intelligence, is introduced as a means of better addressing this problem. Specific types of natural language interfaces are described, together with their strengths and weaknesses. In addition, the application of natural language interfaces to databases is discussed.

WATER RESOURCE PLANNING MODELS

Traditional water resources modeling has focused on the analysis of alternatives. Less attention has been placed on the generation and exploration of alternatives (Loucks et al., 1985). Typically these models are designed to provide quantitative results and are based on the assumption of rational economic behavior. In actuality, rational behavior differs among individuals and is, at best, often unquantifiable.

Water resources modeling also assumed the existence of one or more fixed objectives. In multi-objective modeling, a set of optimal or non-inferior solutions is

generated, usually after much time and expense (Loucks et al., 1985). However, these models do not consider that individuals may want to consider an inferior solution (with respect to the original fixed objectives) as well as the non-inferior ones. These situations arise when decision-makers want to consider additional objectives that are perhaps not quantifiable by the model. This suggests the need to examine more than just a set of optimal or non-inferior solutions for planning models.

One of the largest obstacles in using models as decision-making tools is the lack of confidence and understanding managers have in computer models. Often, analysts and planners do not communicate effectively with one another, prohibiting planners from being actively involved in the model development process. Without this involvement, new questions that arise after initial exploration and development of the model are never addressed, resulting in the final planning tool being unacceptable.

Water resources problems and their impacts are complex and affect a wide variety of individuals and groups of people. In addressing these problems, many complex and comprehensive models have developed. This trend towards ever-more complex models parallels the increase in the memory and the computational capacity of computers and is also affected by the decrease in the cost of powerful computers. These more complex water resource models are more difficult to develop and maintain, require more input data and produce volumes of output that requires extensive analysis to comprehend. A challenge for the future is to understand complex systems so that the models, or more likely a system of more comprehensible sub-models, and the model user interface, becomes more intelligible, manageable, useful and reliable (Quade and Miser, 1983).

The involvement of non-modelers in defining model specifications, model development, model verification and model use will occur only if the interface between the computer, the models and the model users is easy to use and the computer results are easy to obtain and understand. In addition, the interface must make data input and editing easy to perform and make data output easy to manage and comprehend (Loucks et al., 1985; Hendler and Lewis, 1986).

The increasing availability of mini and micro-computers and interactive software has made this goal of user involvement more feasible in recent years (Fedra and Loucks, 1985). One current trend in management and policy making is the increased interactive use of a number of relatively smaller interrelated models designed to be adaptive and responsive to a wide variety of questions that the policy-maker may want to ask. Even if each of the smaller models can only address a few questions, together they can be used to study the more complex structured aspects of a resource management problem (Loucks et al., 1985; Kunreuther and Miller, 1985). A key feature of these decision-aiding systems is the direct involvement of policy analysts in an interactive policy-making process. An important step towards direct involvement of policy makers is the use of highly user-friendly interfaces to models and data.

User-friendliness is defined in a variety of ways. Usually it is defined as characteristics of a computer or of software that allow them to be used without the knowledge of any classical programming languages. Menu-driven programs are an example of one type of user-friendliness. Loucks and Fedra (1985) suggest that user-friendliness also applies to more general aspects of models such as semantic and syntactic consistency (which ensures that the model captures both the intended meaning and syntax), graceful (and instructive) recovery from failures, and a wide

assortment of input-output devices. A user-friendly interface requires the underlying software to be easily understood, well-structured, and compatible with the mental processes of the users (Loucks et al., 1985; Hendler and Lewis, 1988).

This report will describe the development and use of a natural language interface to increase the user-friendliness of a data analysis tool. Because of this interface, the tool is now available directly to water resource managers and decision-makers.

Artificial intelligence tools are becoming increasingly popular for developing user-friendly interfaces compatible with the user's cognitive process. Two of the most common AI tools meeting this need are expert systems and natural language interfaces. Natural language interfaces will be discussed in the following section.

NATURAL LANGUAGE INTERFACES

Natural language interfaces (NLIs) are useful in providing user-friendly access to databases. They do not require extensive training and, therefore, make the data directly available to users with no database or even computer experience. Because they are accessible in ordinary English, they can be used rapidly and accurately in stressful situations or in situations where data may need to be accessed in unanticipated ways (Hendrix and Walter, 1987). The use of natural language processors (NLPs) as interfaces to databases and knowledge bases have been researched since the early 1970's and are still popular topics of study today.

Natural language processing can be defined as the ability of a computer to process the same language that humans use in ordinary discourse. Natural language processing has been studied in six major research areas: (1) natural language interfaces to databases, (2) machine translation (translating from one natural

language to another), (3) text scanning and intelligent indexing programs for summarizing large amounts of text, (4) text generation for automated production of standardized documents, (5) speech systems to allow voice interaction with computers, and (6) tools for developing NLP systems for specific applications. Of these six research areas, developing natural language interfaces to databases is the most active area of research, accounting for over 40 percent of the total activity (Obermeier, 1987).

The main goal in natural language processing is to translate a potentially ambiguous input phrase into a precise form that can be directly interpreted by a computer system. This translation process, called parsing, is performed in many ways. Obermeier (1987) has classified the types of parsers that have evolved into five groups: grammar-based, semantic, pattern-matching, knowledge-based, and neural-network parsers. The first three represent the most active area of interest. These are discussed in this chapter and examples from the literature are given for each. Before they are discussed, however, several terms must be defined.

Language expressions have both syntax and semantics. Syntax refers to the rules governing the order of the symbols. For example, in English an adverb, if used, generally follows the verb to which it applies. For example in "The dog ran swiftly", 'ran' is the verb and 'swiftly' is the adverb. Semantics, on the other hand refers to the intended meaning of the expression. "Mary had a little lamb", could either mean that Mary once owned a small sheep or it could be a description of Mary's evening meal. Computers can easily interpret syntax, but are poor at resolving semantics. Standard language has a prescribed, although sometimes variable, syntax defined by rules. As discussed previously, an expert system is a formal reasoning system using rules, therefore, a natural language processor can be

considered a type of expert system that contains rules for a specific language (Townsend, 1987).

Grammar-based Parsers

Grammar-based parsers are concerned primarily with the syntax of the sentence, that is, the order in which the words appear and their grammatical definition. Grammar-based parsers use a set of rules that describe the types of sentences acceptable for that particular language. For example, two simple rules are:

S ----> NP VP
S ----> VP

where 'S' is the symbol for sentence, 'NP' is the symbol for noun phrase, 'VP' is the symbol for verb phrase, and '---->' is the symbol for 'is defined as'. These rules, called rewrite rules or production rules, define a sentence to have a noun phrase and a verb phrase, or just a verb phrase by itself. These noun and verb phrases are themselves composed of smaller phrases. These phrases can, in turn, be decomposed until only the essential building blocks of grammar remain: individual words. A tree structure can be used to indicate how words in a sentence are related. Figure 2-1 is an example of such a tree; the words at the end of the tree limbs are called terminals and the other symbols in the tree are called non-terminals. Non-terminals can be thought of as symbols which are used to specify grammar.

Grammars used to parse a natural language, such as English, have been categorized into four types (Grishman, 1986). These are, in order of increasing complexity: finite-state, context-free, context-sensitive and Type 0 grammars. Finite-state grammars can only describe simple sentences, such as those given by the sample rewrite rules from the previous paragraph. In practice, English sentences

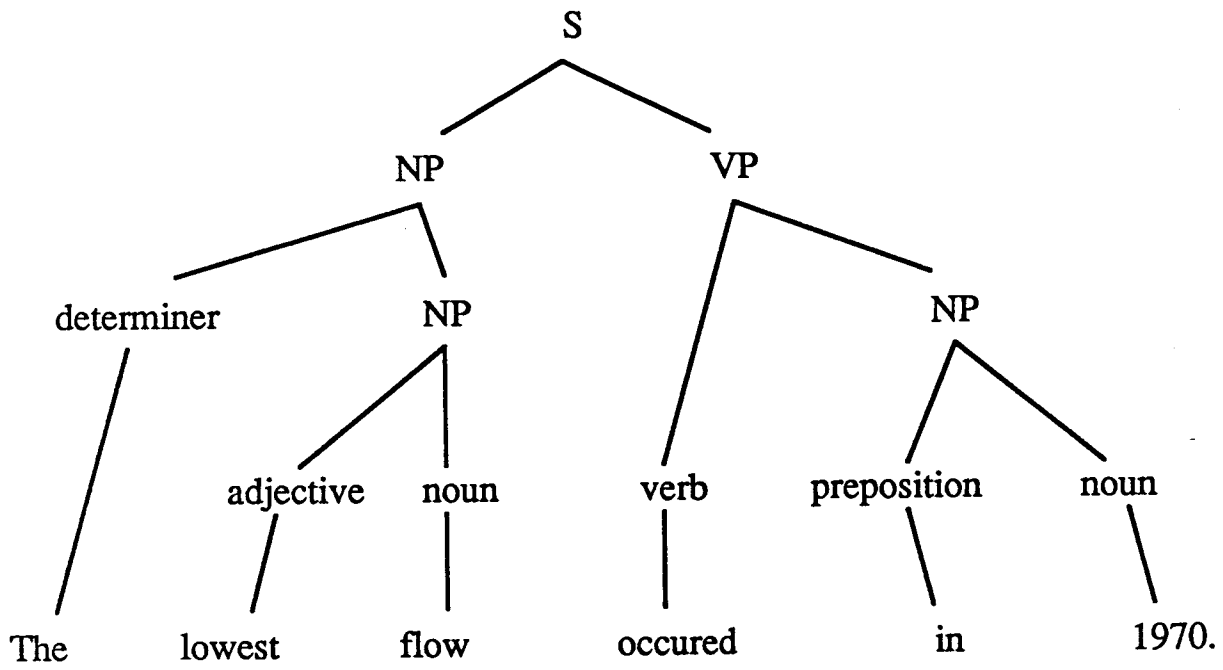


Figure 2-1. Tree Diagram Showing Sentence Structure.

often contain more than one verb phrase or noun phrase, therefore finite-state grammars are not suitable for parsing all of the common sentences found in English.

Context-free grammars require the left side of each rewrite rule to consist of only a non-terminal symbol. For example, the rewrite rule A ----> x means that the non-terminal symbol , 'A', can be replaced by the word, 'x', anywhere it appears in the sentence, regardless of context. In general, context-free grammars are easier to deal with than the more powerful grammars. In some situations, however, constraints will be needed on the context in which a rewrite rule can be applied. For these situations, context-sensitive grammars would be applied.

The context-sensitive grammar allows more than one non-terminal symbol to appear on the left side of the rewrite rule, for example: AB ----> CDE (where A, B, C, D, and E are all non-terminal symbols). This grouping of more than one symbol on the left side means that the context in which 'A' and 'B' are found is important. The most complicated grammar, Type 0, has rules that do not follow any set pattern or requirements, making it very difficult to parse. As a result, Type 0 grammars are not very commonly used in a natural language parser.

Grammar-based parsers use all the words found in a sentence and the phrasing in which the words appeared. They are good for generating natural language text, and determining sentence structure according to the grammar used. Two of the most popular parsers for context-free and finite-state grammars are the augmented transition network, developed by Woods (1970), and the Earley algorithm (Earley, 1970). These two papers describe the theoretical development of these parsers, although they are not applied to a specific system. For natural language systems used as interfaces to databases or expert systems, the semantics

(or meaning) of the sentence, also must be considered in order to correctly perform the request.

Semantic Parsers

Semantic parsers attempt to find the meaning of the sentence, rather than just concentrate on syntax like the grammar-based approach. In semantic parsers, the rewrite rules are stated in terms of "semantic classes" describing the meaning of the word, rather than word classes (ie. verb, noun) like the grammar-based parsers. For example a sentence could be represented by the following:

<SENTENCE> ::= <PERSON> is eating <FOOD>.

In this example, <PERSON> and <FOOD> are semantic classes for which words like "Carla" and "pasta" can be substituted to create a valid sentence. An advantage to semantic parsers is that the size of these semantic classes is generally much smaller than the size of an equivalent word class, resulting in a much more efficient parsing strategy. A disadvantage of using semantic grammar is that it is not easily transferrable from one domain to another.

Two natural language interfaces developed in the 1970's were milestones in making semantic parsers popular. LUNAR (Woods, 1978) was developed for geologists to retrieve and analyze geologic data obtained from the Apollo moon missions. The goal of the LUNAR system was to develop a natural language understanding facility sufficiently natural and complete that the task of selecting the wording for a request would require negligible effort for a geologist to use. The system consisted of three principal components: a general purpose grammar-based parser for a large subset of natural English, a rule-driven semantic interpretation component for transforming a syntactic representation of an input sentence into a representation of what it means, and a database retrieval and inference component.

LUNAR is noted for introducing the augmented transition network (ATN) parser. An ATN parser uses a graph to represent the possible states or semantic categories of a sentence with arcs showing the relationships between these categories. This specific type of natural language parser has become one of the most popular. The LUNAR system had a dictionary of approximately 3500 words and had two databases: a table of chemical analyses with 13,000 entries, and a topic index to documents with approximately 10,000 listings. LUNAR was developed as a prototype system, and although it performed very well in demonstrations, it was never fully implemented and put into operational use. LUNAR did, however, mark a significant step in the direction of fluent natural language understanding.

The other seminal natural language query system, using a semantic parser, is INLAND (Informal Natural Language Access to Navy Data) developed by Hendrix et al. (1978). INLAND is a natural-language interface to a database containing ship data. It was designed to make large, distributed databases directly available to U.S. Navy decision-makers, while freeing technicians from increasingly tedious details. Some sample questions that INLAND can answer are:

> What is the name and location of the carrier nearest to New York?
> How long is the Kennedy?
> To what class does the Soviet ship Minsk belong?
> Which U.S. Navy convoy escorts have inoperative sonar systems?

Within INLAND, syntax and semantics were combined in a single semantic grammar. INLAND was part of a larger natural language interface, called LADDER (for Language Access to Distributed Data with Error Recovery) which provided access to many other databases. LADDER was developed to be transportable from one database to another regardless of the underlying database structure. Many important human engineering features contributed to the success of the LADDER

system. Some of these features were the ability of the system to deal with incomplete input, allowing the user to extend the linguistic coverage at run time and highly user-friendly functions for modifying and debugging application languages.

Pattern-Matching Parsers

The pattern-matching parsers include some of the earliest parsers developed. They look for a linguistic pattern in a sentence without using explicit grammatical rules. When processing a sentence, this type of parser attempts to match the input with a fixed number of patterns. If a match is found, the system performs a specified action. Pattern-matching parsers are popular as interfaces to databases because database commands can usually be decomposed into patterns and keywords.

Two of the best known pattern-matching programs, ELIZA (Weizenbaum, 1966) and DOCTOR (described by Winston, 1981), were not database interfaces but rather were designed to simulate interaction with a psychologist. Both consist of a set of patterns with each pattern having a number of replies associated with it. When a particular pattern is matched, the program selects one of the given set of replies and makes any necessary substitutions in that reply. A portion of a sample dialogue from DOCTOR might be:

DOCTOR: Speak up!
User: I am worried about the weather.

DOCTOR: How long have you worried about the weather?
User: Since my mother passed away.

DOCTOR: Tell me more about your family.
User: They are all into computers these days.

DOCTOR: Do machines frighten you?
etc.

In these programs, the appearance of understanding is derived from a clever use of keywords rather than from genuine analysis and appreciation of what is said (Winston, 1981).

One of the first natural language interfaces to a database was BASEBALL (Green et al., 1961) which answered questions about the games played during one season of the American League. This system read a sentence and then formed a specification list. This list corresponded to a list of keywords which were looked up in the system's dictionary and then translated into a database query. BASEBALL's existence proved that natural language systems were possible and it started one of the major lines of research, pattern-matching, in NLP today.

A natural language interface to DOS, using a noise-disposal parser similar to BASEBALL, is described by Lane (1987). A noise-disposal parser scans the user's request and finds keywords upon which the action is based while non-essential words (noise) are ignored. Lane's NLI, implemented in Turbo Prolog, collects the tokens (or words), finds the keywords and then performs the command. This system even provides facilities to correct misspelled input in some situations.

INLET, the topic of this report, uses an approach similar to Green (1961) and Lane (1987) and is applied to a water resources database. The database contains streamflow data along with optimal reservoir operating policies generated from a drought management expert system (Palmer and Holmes, 1988; Palmer and Tull, 1987). INLET uses a noise-disposal parser to find keywords, match them with a command pattern and then perform the command. Often, a noise-disposal parser requires a strict sentence format (an ordering of the keywords), but it will accept a wide variety of sentences as long as the necessary keywords are present (Schildt, 1987). In INLET's case, however, the keywords may come in any order.

Considerations and Criticisms of Natural Language Interfaces

Hendrix, the author of INLAND, and Walter describe general technical considerations involved in designing a natural language interface (Hendrix and Walter, 1987). Some of the important considerations applicable to INLET are habitability, verifiability, resilience and performance. Habitability is the characteristic of a system which allows the user to express himself through English requests that come readily and comfortably to mind. Verifiability means that the user can verify that the NLI's interpretation of the request agrees with what the user meant. The NLI also needs to be resilient, or recover gracefully from anomalies in the user's request. Finally performance considerations are important to assure the system responds quickly and efficiently to the user.

Templeton and Burger (1986) describe the pitfalls and necessary requirements for developing a successful NLI to a database. Their research is based on their experience in developing EUFID, the End-User Friendly Interface to Data management (Templeton and Burger, 1986). EUFID was designed to be both application and database management system (DBMS) independent. This was an overly optimistic goal and the system enjoyed only limited success. The necessary requirements outlined by the authors which are relevant to this research are in the area of "language problems", or problems understanding the natural language question as opposed to technical database issues or coding issues. They found that a NLI needs to respond promptly to requests, either with an answer or with a helpful message describing why the request could not be interpreted. Another problem was encountered in defining the scope of reference in the request. An example from INLET might be:

> "What was the mean flow at Site 1 for June?",
> "What was the mean for June between 1960 and 1970?",

"What was the skew?"

In this example, the second question refers back to the first question in which Site 1 was specified; the third question could refer to all of June or just the years 1960 to 1970. This problem is difficult for most NLIs to decipher. The authors of EUFID suggest confining the scope to a subset of the database and then only addressing questions to the subset. There are many other issues discussed and suggestions made in this paper, most concern making the interface transportable and database structure independent.

Some criticisms of natural language interfaces have been cited by Simmons (1986). He discussed a then-recent test comparing the success of using a NLI with the success of using a Standard Query Language (SQL). Users were required under various experimental conditions to formulate a series of questions in either SQL or English. The SQL queries resulted in 46 percent correct answers; the NLI queries only 22 percent. Simmons explained the reason for this difference as being due to the feedback (or guidance) available for SQL and the lack of feedback provided by the NLI. With no feedback from the NLI, users were left to their own devices, usually random paraphrasing of the question. The NLI used in the test, however, was not fully debugged; general studies show NL questioning to be about 70 percent satisfactory (Simmons, 1986). This test clearly showed the importance of a good feedback or help system in order for a NLI to be successful.

SUMMARY

This chapter has defined the need for more user-friendly models to encourage direct involvement of decision-makers. Natural language processing was described and some example systems were discussed. Natural language interfaces are very useful as database interfaces when the users are knowledgeable about a task in a domain whose scope is limited and when intermittent use inhibits command

language training (Sneiderman, 1986). This is certainly the situation in which INLET is designed to be used. Most of the considerations outlined here have been addressed by the INLET system and will be discussed in Chapter 4. A summary of the Seattle Water Supply System will be presented in Chapter 3 along with a description of the drought management expert system.

CHAPTER 3

SEATTLE WATER SUPPLY SYSTEM

The Seattle Water Department (SWD) provides direct service to 541,000 Seattle residents and is a wholesaler to 549,000 residents of King County (SWD, 1986). Water is taken from two major sources, the Tolt and Cedar Rivers, both of which originate in the western Cascade Mountains. In recent years there has been a growing concern that the demand for water in this region has approached the safe yield of the supply system. This concern has, in turn, lead to increased interest in the proper management of these resources and the development of operating strategies for the system during droughts.

This chapter describes the water supply system of Seattle and the uses to which the water in this system is placed. Estimates of the safe yield of the system are provided with a discussion of the water shortage response plan that has been established. Previous models used for planning are briefly described, together with an expert system that has been developed for managing the water supply during periods of low flows. The chapter concludes with a description of the information generated from previous studies that is used in the INLET system.

SEATTLE WATER DEPARTMENT SYSTEM

Figure 3-1 illustrates the Seattle Water Department's service area and its major reservoirs. The Cedar River drains approximately 190 square miles and receives about 35 inches of precipitation per year. The Cedar River has served as a source of water for Seattle since 1901 when a diversion dam was constructed at Landsburg, Washington. In 1905, a wooden crib dam was constructed immediately

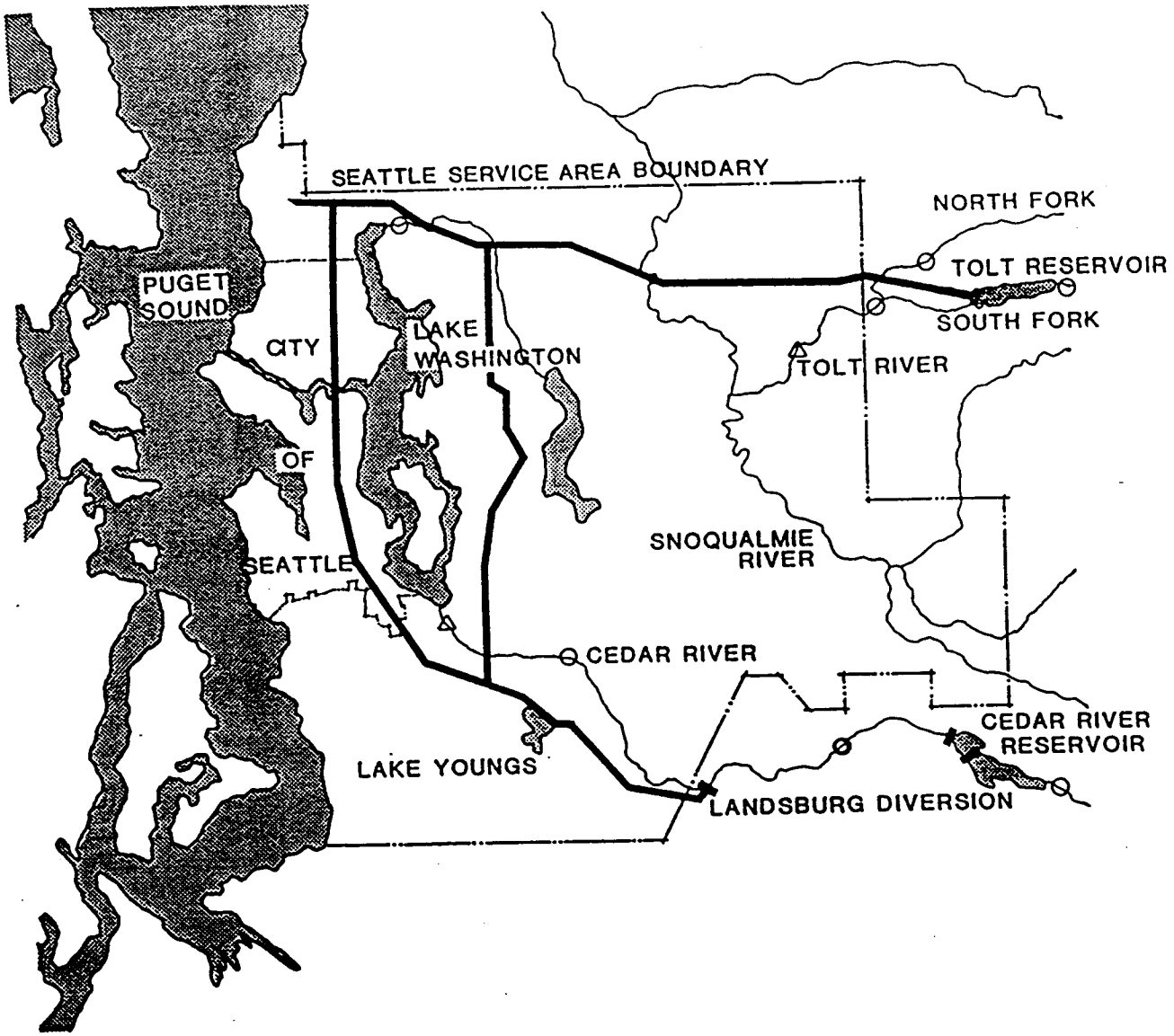


Figure 3-1. SWD's Service Area and Major Reservoirs.

downstream of Chester Morse Lake. In 1914, a masonry dam was built approximately one mile further downstream. Because the masonry dam is sited on top of a glacial moraine, there is a major problem with seepage. Storage can not be maintained at the full capacity of the masonry dam because of fears of a moraine failure that would result in dangerous flows downstream. This has, in fact, occurred in the past resulting in the loss of several lives. Currently, the crib dam (which is upstream from the masonry dam) is used to retain water in a region not subject to seepage. A small amount of hydropower is also generated on the Cedar River. Currently, water from the Cedar contributes about two-thirds of the total water supply. In addition, the Cedar River supports the largest sockeye salmon run in the continental United States. The run was established in 1937 and made possible after the Corps of Engineers constructed shipping locks in 1916. These locks provided a direct connection between Lake Washington and the Puget Sound.

The second major source of water is the South Fork of the Tolt River. An earthen dam was constructed on this river in 1963. This reservoir captures water from approximately nineteen square miles. The Tolt serves the remaining one-third of the SWD water supply. Consideration has been given to expanding the Tolt system by developing a diversion dam on the North Fork of the Tolt. The river supports a wide variety of fish, with the predominant species being steelhead trout. Instream flow requirements have been established for the North and South Forks of the Tolt as well as its main stem.

The established minimum instream flows requirements limit diversions from each river. These instream flow requirements exist to maintain water for fisheries, hydropower generators, and recreation usages. The storage capacity of the Tolt system is 57,000 acre-feet. The average annual inflow into the system is 126,000

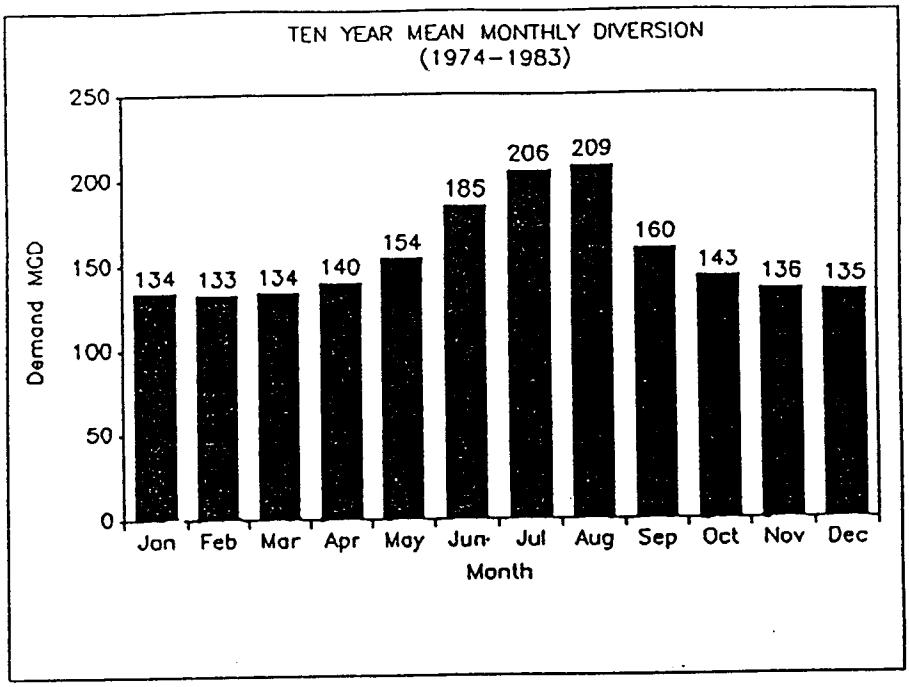
acre-feet. The Cedar contains approximately 39,200 acre-feet of active storage and 39,200 acre-feet of dead storage behind the crib dam.

SYSTEM YIELD AND DEMAND

Although averaging over thirty inches of rainfall annually, the Seattle area is susceptible to droughts. SWD estimates their average annual demand in 1985 at 178 million gallons per day (mgd). Of that total, approximately 79% goes to direct or indirect services, 11% goes to non-revenue sources (such as system maintenance, water for parks and the flushing of Green Lake), and the remaining 10% is attributed to system losses. There is a large temporal variation in demand, both seasonally and yearly (Figure 3-2). The rate of increase of demand is estimated at 2 mgd per year.

SWD estimates that their "safe yield" is 169 mgd. (In this report, safe yield is defined as the maximum volume of water that reliably can be taken from the water supply system while meeting all of the physical, hydrological, and operational constraints of the system.) This yield results from the combined releases of the Tolt and Cedar Rivers. The figure of 169 mgd is based upon a perceived 98% reliability of meeting all demands over a fifty year hydrologic record. Until new sources are developed, little excess capacity exists to meet unusually high demands or low supply situations. The question of reliability is an important and sensitive issue.

For successful operation during the summer, reservoir storage levels must be near capacity after the spring snowmelt. Average rainfall during July and August is 1.8 inches. Thus, the system also depends on autumn precipitation to refill its reservoirs. Unusual climatic events, such as the drought that occurred in 1987, cause system storage to decline to levels that require water use restrictions. Although not commonly used, these restrictions are the only management option available for coping with short-term water shortages.



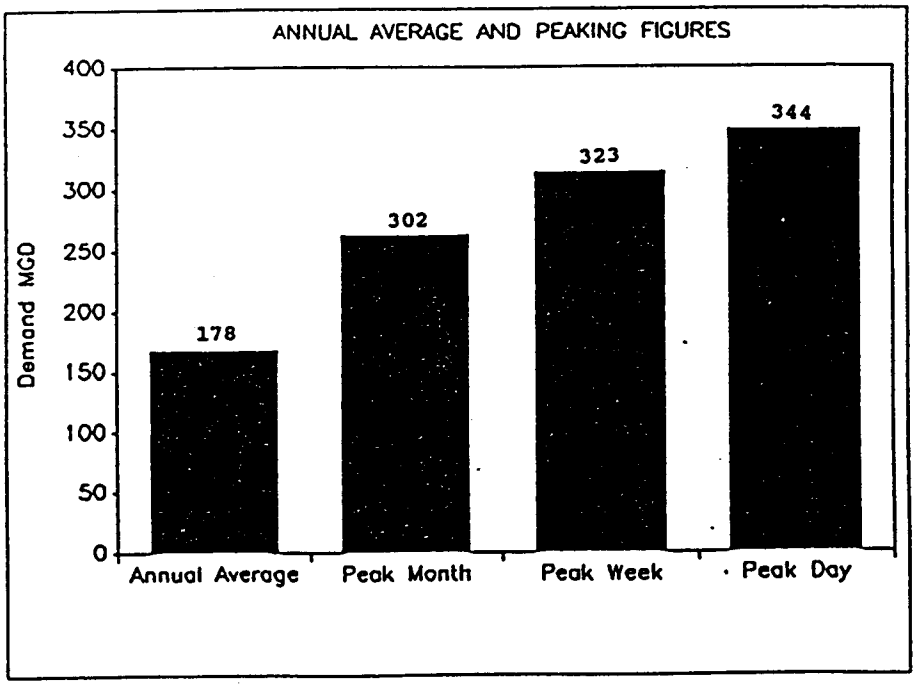


Figure 3-2. Average Seasonal and Peak Demand for SWD.

WATER SHORTAGE RESPONSE PLAN

In order to guarantee an orderly response to any water shortage, the SWD developed the Water Shortage Response Plan (WSRP) which addresses problems related to the 1-in-50 year drought event (SWD, 1986). The objective of this plan is to maintain essential services while minimizing the net economic loss during a drought event. The WSRP envisions two types of shortages: a summer shortage and a fall shortage. Each type of shortage plan consists of a multi-stage conservation plan with progressively higher stages initiated as serious conditions develop. Summer shortages result from climatic and hydrologic conditions that cause system reservoir levels not to be refilled by late spring. The fall shortage scenario results from low fall precipitation that is insufficient to replenish the system storage after summer peak use. Table 3-1 provides a summary of the shortage plan.

Conflicts in operational objectives between the City of Seattle, the Army Corps of Engineers, the Washington State Department of Fisheries, and the Washington State Department of Ecology are common during periods of low flow. These conflicts result from the different objectives the agencies have for water usage in the Tolt and Cedar watersheds. Figure 3-3 illustrates the summary and conceptualization of the decision process used for drought planning. Decisions are made in a formal, if ad-hoc manner, attempting to weigh conflicting objectives of the agencies. At times, only a limited amount of quantitative data are available to all agencies and the precise impacts of their decisions often are not known.

Table 3-1. Summary of Summer Shortage Response Plan

Stage 1 - Minor Shortage Potential

Cause for implementation:

1) Total system storage not filled by June 1.
2) Reservoir storage will not be replenished before the peak demand season.

Water system actions:

1) Eliminate non-essential water uses.
2) Reduce Green Lake inflows from 4 MGD to 2 MGD.

Customer actions:

None

Estimated total savings:

3.0 MGD

Stage 2 - Moderate Shortage Potential

Cause for implementation:

1) Total system storage is predicted to fall below level required to meet demand during a 1-in-50 year drought.
2) System inflows continue to be low.
3) Predicted drier than normal weather.

Water systen actions:

1) Adjust Cedar and Tolt releases to increase stored water.

Customer actions:

1) Voluntary reduction in outdoor water use.

Estimated total savings:

7.7 MGD

Stage 3 - Serious Shortage

Cause for implementation:

1) Total system storage is predicted to fall below level required to meet demand during a 1-in-50 year drought
2) System inflows continue to be low.
3) Predicted drier than normal weather.

Water systen actions:

1) Reduce Cedar and Tolt River flows to critical streamflow levels.

Customer actions:

1) Mandatory reduction in outdoor water use.

Estimated total savings:

20 MGD

Table 3-1. Summary of Summer Shortage Response Plan (continued)

Stage 4 - Severe Shortage

Cause for implementation:

1) Total system storage is predicted to fall below level required to meet demand during a 1-in-50 year drought.
2) System inflows continue to be low.
3) Predicted drier than normal weather,
4) Entering end of peak use season.

Water systen actions:

1) Adjust Cedar and Tolt stream flows to critical levels.
2) Request Army Corp of Engineers to reduce flows for Chittenden Locks.

Customer actions:

1) Mandatory reduction in outdoor water use.
2) Voluntary reduction in indoor water use.
3) Voluntary reduction in commercial uses.

Estimated total savings:

21.1 MGD

Stage 5 - Critical Emergency

Cause for implementation:

1) Customer demands and system pressure requirements cannot be met.

Water systen actions:

1) Reduce Cedar and Tolt River flows to critical streamflow levels.
2) Request Army Corp of Engineers to reduce flows for Chittenden Locks.

Customer actions:

1) Water rationing to gain a 40% reduction in overall water use.

Estimated total savings:

57.4 MGD

Source: Seattle Water Department

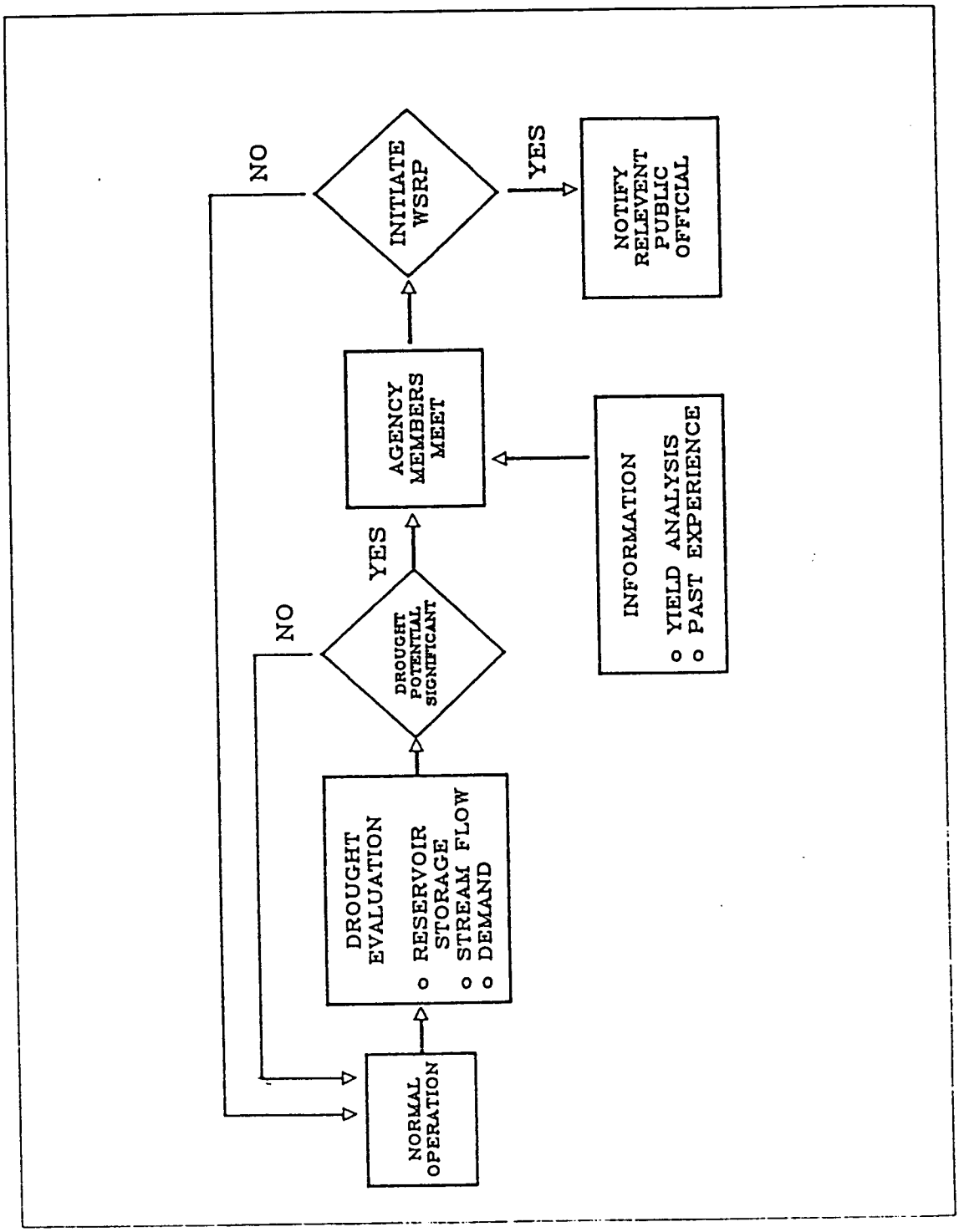


Figure 3-3. Flow Diagram of Current Planning Methodology.

PREVIOUS MODELING EFFORTS

The SWD has maintained a continuing effort to improve its understanding of the hydrology and operation of its system. A result of this effort has been the development and refinement of system management computer models. Three models are in current use, the Water Resources Management Model (WRMM), the Seattle Water Supply Expansion Management System (SEMS) and the Seattle Water Department Integrated Drought Management Expert System (SID).

WRMM is a batch simulation model developed in early 1980's that runs on a mainframe computer (URS Corporation, 1981). It is currently used by SWD to project annual water yields and analyze the probability of a water shortage in any given year. The model operates on a monthly time-step. As a simulation model, it does not provide operating policies, it only reports what would occur if drafts from the reservoir systems were taken at a specific rate. Although the model is somewhat difficult to use and requires a high level of training to operate and interpret the results, it is the model that has been used for the longest period of time by the SWD.

SEMS is an interactive linear programming model that was developed to operate on a personal computer (Palmer and Johnston, 1984). This model develops optimal operating policies for a given system configuration of the water supply. SEMS operates on a weekly time-step, providing optimal operating policies for a sixteen to twenty week period into the future. The user can select between five potential operating objectives including the maximization of yield and the minimization of economic losses to the system. The user operates the model by indicating the starting time period (week of the year), the anticipated demands, the system storage levels, and the hydrologic sequence desired. The user can choose between any historical record or streamflow forecast.

SID is a recently developed expert system for drought management (Palmer and Tull, 1987; Palmer and Holmes, 1988). The system is composed of an expert system, a database management system, and computer graphics. These three components have been integrated into a complete drought management system. The expert system incorporates operator experience and intuition using a rule base developed through interviews with management personnel from SWD. SEMS was used to create a large database of operational experiences and examples. Over three thousand executions of the SEMS model are contained in the database. The results can be obtained through the database management system and displayed using the graphics software.

DEVELOPMENT OF THE INLET DATABASE

INLET represents a natural progression in the computer models that have been developed to aid in obtaining, analyzing, and understanding information and data required for developing municipal operating strategies. Its purpose is to provide the user with easy and direct access to all relevant hydrologic and operational data with a minimum of training. Whereas WRMM requires extensive training and experience, INLET provides the novice user with immediate data retrieval abilities. SID requires the user to proceed through a formal and complete drought management evaluation, INLET, however, allows the user to access whatever information is desired, in whatever order is most appropriate. INLET also provides extensive statistical and graphical presentation of streamflow data that was not available in previous system modeling efforts.

The development of INLET and its functions and capabilities are described in detail in the following chapter. The remainder of this chapter is devoted to a description of the database to which INLET provides access. The data used in

INLET is of two basic types: hydrologic data and system management data. The hydrologic data consists of monthly streamflow records for seven sites in the water supply catchment area. Three of these sites are on the Cedar River, one is the net inflow to Lake Washington (inflow minus evaporation), and the remaining three are flows on the Tolt River: the South Fork, the North Fork, and the main stem.

The management data was derived by interactive executions of the SEMS program. A position analysis (Hirsch, 1977) was developed using SEMS. In this procedure, past streamflow data serve as surrogates for potential future inflows. State variables (time of year, storage levels in the reservoirs, system capacities, and demand) are defined and the system is simulated over a prescribed period. Variables of interest are recorded and the procedure is repeated for the next period.

Although Hirsch performed his analysis with a simulation model, SEMS is an optimization model, as indicated previously. SEMS operates on a weekly time period, incorporates physical and operational constraints and optimizes the system's operation. Two objectives are used for the information contained in the database: maximize system yield and minimize the economic loss associate with deficits from a specified (and time dependent) target. For this system, the planning period is four months with each year being an independent event. Streamflow data exists for nearly fifty years for the primary sites of interest in the system, making this an especially attractive approach.

For the first objective, estimating the system yield, the constraints include continuity at both reservoirs, instream flow requirements and continuity for the moraine aquifer in the Cedar system. The Cedar system is more complex because of this aquifer. The aquifer is recharged by seepage from Masonry Pool and returns water to the Cedar River further downstream. Both events occur at unknown rates

which the model approximates (Palmer and Johnston, 1984). Bounds exist for storage levels on all reservoirs, pipeline capacities, instream flow requirements for the Tolt and Cedar Rivers, and the Lake Washington elevation.

The second objective requires several additional constraints to meet water use implementation requirements. Upper bounds on each stage of WSRP restrictions represent the maximum reduction in water use that is possible for each stage. A piece-wise linear objective function approximates the economic losses associated with the implementation of water use restrictions (SWD, 1986). Additional constraints limit stage increases to one per week, thus preventing staging from skipping two or more levels in one week.

SEMS was executed using the historical record (1929-1975) for a variety of reservoir storage levels, starting dates, and system demands. System yield was calculated for each streamflow record for an initial storage of between 10 and 100% of capacity (by units of 10%) and for June through October. This required 2,350 runs of the model. The results of the yield analysis identified configurations that result in economic losses for specific demands. The minimum economic loss, and its associated operating policy, was calculated for the appropriate configurations for base level demands of 170, 180, and 190 MGD. This resulted in approximately 600 additional runs of the model.

The execution of SEMS requires approximately three minutes of central processing unit (CPU) time on an IBM/AT using XA, a linear programming software package (Sunset Software, 1987). Execution time for calculating economic loss are less predictable, requiring between three and six minutes. Approximately 200 CPU hours were required for all linear programming runs. This task was simplified by writing software to automate this process.

A separate database system, created to store this information, was written using codes from the Turbo Pascal Database Toolbox (Borland International, 1985). The database stores the following information:

Record code information
Starting week
Base level demand
Initial storage
Yield array
Economic loss array
Total number of drought years
Drought years
Water Shortage Response Plan staging data for drought years

The database stores the record code, week, and base level demand as a series of characters, the yield and loss arrays as 46 element arrays of real numbers, the drought years as an array of integers, and the Water Shortage Response Plan (WSRP) staging data as a two-dimensional array of real numbers.

SUMMARY

The data described above provides the database from which INLET obtains information about the hydrology and operational policies of the SWD system. Using this database INLET also performs statistical analysis, ranking, plotting, and other evaluations which are not available in the expert system or any other analysis tool currently available to the SWD. This direct access to and statistical analysis of streamflow data upon which other models are based is a valuable function performed by INLET.

INLET is unique among the models and analysis tools that have been developed for the system because of the ease of access and range of analysis made available. The background and program development of the INLET system are the subject of Chapter 4.

CHAPTER 4

INLET, A NATURAL LANGUAGE ENVIRONMENT

An Interactive Natural Language Environment, INLET, was developed to provide user-friendly access to streamflow and optimal reservoir operation policy data. This system has two components: a natural language interface and a menu-driven interface. Both components provide direct access to the data and provide statistical and plotting capabilities. Both of these components are easily accessible from the other. INLET was written in Turbo Prolog (Borland International, 1988). Streamflow data and operating policies relevant to the Seattle Water Department were incorporated into INLET to provide a case study. This chapter describes INLET and provides an introduction to the Prolog programming language. This is followed by a description of the natural language interface and a description of the menu system.

INTRODUCTION TO PROLOG

During the past decade, two languages have gained prominence for the development of artificial intelligence (AI) applications. These languages are LISP (for LISt Processing), and Prolog (for PROgramming in LOGic). LISP is one of the oldest computer languages, having been developed in the 1950's at approximately the same time as Fortran. Prolog is a much newer language, developed by Alain Comerauer at the University of Marseille-Aix, France, in 1972. Prolog is similar to LISP in that it is a symbolic language and it was developed specifically to solve AI-related problems. Prolog differs from LISP because it has a built-in database and has a simpler syntax.

In the early 1980's, Prolog had established itself as the AI language of choice in Europe while LISP emerged as the preferred language in the United States. In

1981, researchers participating in the Japanese "Fifth-generation" computer effort announced the use of Prolog as their fundamental programming language.

Since 1981, there have been fundamental changes in the Prolog language. Many implementations of the language are commercially available with supporting software. Interest at universities in the U.S. has increased and the number of textbooks, reports, and journals devoted to Prolog are also increasing. The structure of the language has also formalized to a defacto standard of "Edinburgh Prolog." Although variations of this standard exist, most software meets the basic standards of Edinburgh Prolog and supply the user additional features.

Prolog differs greatly from standard engineering computer languages such as Fortran or Pascal. Programming languages can be described as either procedural or descriptive. In procedural languages, program execution is sequential; it follows the order of the commands found in the source code. Fortran is an excellent example of a procedural language. Prolog is a descriptive or "data-driven" language. A Prolog program is essentially a database and a series of rules for analyzing the data. The control of execution is determined by the specific data in the database, not by a fixed algorithm. This allows the execution procedure to be dynamic, varying with the problem definition.

Prolog supports recursive functions. Recursion is the process of a function calling itself or executing itself. This feature will be described later in this section with an example. This is a capability that is not supported by procedural languages. Prolog also supports symbolic processing. Symbolic processing allows a problem to be solved by strategies and heuristics for manipulating symbols rather than using defined algorithms. For example, the fact that Bill likes Mary can be expressed as a valid fact in a Prolog program by:

likes(bill, mary).

This type of relationship is more difficult to express as a fact in non-symbolic languages, such as Fortran, Pascal or C.

Prolog can make deductions. Given the facts:

"Bill likes Mary."
"Mary likes Sam."

and the rule:

"Bill likes a person **if** Mary likes a person.,"

Prolog can deduce that Bill likes Sam. These facts and rules can be expressed in Prolog by the following statements:

```
likes(bill, mary).
likes(mary, sam).
likes(bill, Person):- likes(mary, Person).
```

The left-hand side of the rule is called the head, the characters ":-" implies the word "if." All symbols to the right of the character ":-" are called the body. The head is a fact that is true if the conditions contained in the body are true. The if symbol separates the head of the rule from the body; it can be the characters ":-" or the word "if." Words in lower case are specific constants. The words that are capitalized (i.e. Person) can be thought of as variables. The rule,

likes(bill, Person):- likes(mary, Person)

can be interpreted as bill likes any Person if Mary likes that Person.

PROGRAMMING IN PROLOG

A Prolog program consists of three components: (1) data or facts, (2) rules, which specify relationships between the data, and (3) a goal or a group of goals. A *predicate* is a function with a value of true or false that expresses a property or relationship. In INLET, the streamflow data is contained in the predicate

"flow(year,month,streamflow)." Data and rules contain predicates. A *clause*, in Prolog, is a fact or a rule that ends in a period. Some particular clauses of the flow predicate are:

```
flow(1929,october,444).
flow(1930,october,56).
flow(1931,october,225).
```

These clauses define the streamflows for October in the years 1929 through 1931.

Rules in Prolog are true if all of the premises are true. If any premise fails, the rule fails at that premise. A rule which determines "low flows" might be "A flow for Year and Month is low **if** the flow is less than 60 cfs." This rule can be written in Prolog as

```
lowflow(Year, Month, Flow) :-
        flow(Year, Month, Flow),
        Flow <= 60.
```

This rule states, "To prove that a flow is low, first prove that the flow for a particular year and month is less than 60 cfs." Using the above streamflow data, the clause "lowflow" would find that a low flow occurred in October of 1930.

After establishing data and rules, the next step in developing a Prolog program is to form a goal or query for the program . A sample question for the streamflow data is "What is the streamflow for October in 1930?" This question would translate to the Prolog goal of

```
flow(1930,october,Value).
```

This query looks very similar to a fact clause except the variable, or object , "Value" is capitalized while the other objects are not. This is because "1930" and "october" are constants, or known values, but "Value" is a variable.

To find a solution to this query, Prolog begins with its first fact and looks for a match. The first fact, flow(1929,october,444), is evaluated but rejected because

the first argument, 1929, does not match 1930. Next, the fact, flow(1930,october,56), is evaluated and the variable "Value" is bound to "56" since all the other objects (the year and month) are equal. Prolog then continues to seek other solutions for this clause. However, no other solutions will be found since only one streamflow for October, 1930, is contained in the facts.

This example illustrates an important characteristic of Prolog that differs from other programming languages. Prolog has no assignment statement; objects or variables obtain values when bound to constants in facts or rules. Until a variable is assigned a value it is said to be "free"; when it is assigned a value it is said to be "bound." The value remains bound while a goal is pursued. When a new goal is requested, the program begins the process of binding variables once again. In Prolog, variables can not be used to store information.

When Prolog searches for a solution to a goal it may pursue a series of rules that fail to instantiate the goal. When this occurs, Prolog automatically retreats to the most recent goal that has an untried alternative, and using that alternative, searches its data again. This process, in which the program retraces a sequence of inferences, trying to find another path to a solution, is called backtracking. In the streamflow example, the first clause that was evaluated was the clause for the year 1929, which failed. In this case, Prolog found a clause that matched the goal, then bound "Value" to "444" and checked the year and month. When they did not match, Prolog backtracked by freeing Value from "444" and proceeded to the next clause.

Prolog has another distinctive feature: recursion. A recursive procedure is a procedure that calls itself. For example, finding the factorial of a number, N, can be written recursively. The procedure for determining the factorial of a number can be written as:

If N is 1, the factorial is 1.
Otherwise, find the factorial of N-1, then multiply it by N.

This procedure can be used as follows: To find the factorial of 3, you must find the factorial of 2; to find the factorial of 2, you must find the factorial of 1. From the first clause in the procedure, the factorial of 1 equals 1, so now you multiply it by 2 to get the factorial of 2, then multiply that by 3 to get the factorial of 3. In Prolog this procedure would be

```
factorial(1, 1).
factorial(X, FactX) :-
        Y = X - 1,
        factorial(Y, FactY),
        FactX = X * FactY.
```

When Prolog executes factorial(3, Answer) it will try the first clause, "factorial(1,1)," and fail. It then moves on to the next clause and binds X to 3 and searches for a solution for Answer. Next it sets Y to 2 and then calls factorial again with the first object bound to 2 and the second object still unbound. This procedure repeats until factorial is called with X bound to 1. At this point factorial succeeds with FactY being bound to 1. Now the factorial which was called with X bound to 2 can calculate FactX and succeed. When it succeeds the original factorial can calculate FactY and that value is bound to Answer.

Recursion has three main advantages over conventional iterative loop structures, such as the "do loop" found in Fortran. It can express algorithms that can not conveniently be expressed any other way. It is logically simpler than iteration. Recursion is the natural way to describe any problem that contains within itself another problem of the same kind.

An example of recursion is the process of checking whether a symbol is a member of a list. One may approach this problem by first checking to see if the

symbol is at the head of the list. If it is not, check if it is a member of the remaining list. This process requires breaking this list into two sets: the first item and all the remaining items. Since the tail (which is everything but the first item) is a list, checking if the symbol is in the tail is the same problem as the original, only with a shorter list. In Prolog, a list is represented by a sequence of objects separated by commas and is contained within square brackets. An example of a list is [Ron, George, Oliver]. Lists can also be represented as just two components in brackets separated by a vertical line:

[Head|Tail]

The first element of the list is Head and the remainder of the list is contained in Tail. In our list example, "Ron" corresponds to the Head and "[George, Oliver]" corresponds to Tail.

In Prolog, the procedure to check if an element is a member of a list could be coded as

```
member(Element,[Element|_]).
member(Element,[_|Tail]):-
        member(Element,Tail).
```

The first clause succeeds if Element is the first item in the list. If not, Prolog continues on to the second clause and calls member again with Element and the tail of the list.

IMPLEMENTATION OF PROLOG

A large number of implementations of Prolog are available for personal computers. The implementation chosen for this research is Borland's Turbo Prolog (Borland International, 1988). This version of Prolog is the most widely sold implementation for personal computers. The developers of this software have concentrated less on making this version totally compatible with the "Edinburgh

Prolog" standard and more on providing supplemental programs and supporting software to improve the ease with which sophisticated programs can be written.

All Turbo Prolog programs have three components: (1) names and structures of objects involved in the problem, (2) names of relations known to exist between the objects, (3) facts and rules describing these relations. These three components correspond to the three basic parts of a Turbo Prolog program: the domains section, the predicates section and the clauses. This three-part structure is unique to Turbo Prolog. Edinburgh Prolog does not use the same program partitioning. An example program will be used to show what these three parts of a program do and how facts and rules are incorporated.

The domains section contains the declaration of the types of arguments allowed for the predicates. Standard Prolog does not require a domains section. Under the predicates section, predicates are declared by stating their name and the domains of their arguments. Clauses are facts or rules corresponding to one of the declared predicates.

Figure 4-1 shows simple Turbo Prolog program. The domain indicates the variable types that are used in the predicates. These domains state that flows are real numbers, a year is an integer, and a month is a symbol (or word). The predicate section lists all the non-internal predicates to be used in this program, "sumflow" and "flow." Prolog has many internal predicates, such as "write(symbol)" which displays symbol on the screen. These internal predicates are similar to intrinsic functions in Fortran and do not need to be defined by the program.

The next section denoted as the goal section is optional. If the goal section is not present, Turbo Prolog responds with a prompt and waits for a goal to be entered

```
domains
        flows    = real
        flowlist = flows*
        year     = integer
        month    = symbol
        statistic = symbol
        command  = symbol

predicates
        summer(year,flows)
        flow(year,month,flows)

goal
        summer(1950,Sum),
        write("The sum of the 1950 summer flows is ",Sum).

clauses

        summer(Year,Value):-
                flow(Year,june,A),
                flow(Year,july,B),
                flow(Year,august,C),
                Value = A + B + C.

        flow(1950,january,278).
        flow(1950,febuaury,321).
        flow(1950,march,350).
        flow(1950,april,217).
        flow(1950,may,197).
        flow(1950,june,56).
        flow(1950,july,61).
        flow(1950,august,35).
        flow(1950,september,94).
        flow(1950,october,189).
        flow(1950,november,312).
        flow(1950,december,253).
```

Figure 4-1. Sample Prolog Program.

interactively. The goal of this program is to calculate the sum of the summer flows for 1950 and then display the result.

The clause section contains the definition for "sumflow" and the "flow" clauses containing the streamflow data. "Sumflow" finds the sum of the summer flows by searching through the flow clauses, matching Year and Month. When each the three flow clauses have been found, A,B, and C are bound to the three flows corresponding to the June, July and August flows, respectively. Sum is then bound to their sum and returned. The first goal has now been satisfied. Next Prolog displays "The sum of the 1950 summer flows are" and then the answer, Sum.

INLET'S NATURAL LANGUAGE PROCESSOR

INLET's natural language processor reads an input sentence and translates it into a command the computer can execute. It is also responsible for answering the query in a full sentence, providing it is not a plot. INLET uses a noise-disposal parser to scan the input sentence, evaluate keywords and dispose the non-essential words (noise). The keywords are then matched to a pattern and the command is performed. This type of parser is popular as an interface to a database, because database commands can usually be decomposed into patterns composed of keywords.

INLET recognizes words from the following keyword groups:

<command> <statistic> <site> <month> <year 1> <year 2>.

The words in the brackets are the names of the word classes. The brackets indicate that the keyword inside is optional. The words recognized in each category are listed in Table 4-1 along with their default values. The keywords can appear in the

Table 4-1. Words Recognized by INLET

Commands	Statistics	Sitename	Month	Year 1	Year 2
what	all	Site 1	January	1929 -	1929 -
plot	mean	Site 5	February	1976	1976
list	standard	Site 7	March		
help	skew	Site 8	April		
policy	lowest	Site 11	May		
	highest	Site 15	June		
	minimum	Site 16	July		
	maximum	inflow 1	August		
	storage	inflow 2	September		
	average	inflow	October		
		Cedar 1	November		
		Cedar 2	December		
		Cedar 3			
		Lake Washington			
		North (Fork Tolt)			
		South (Fork Tolt)			
		Main Stem (Tolt)			
		demand			
("what") ("none")	("all")	*	("none")	("none")	("none")

words in () indicate default values

* default is determined from previous command

sentence in any order. Some example sentences that can be answered are:

> What is the mean flow at Cedar 1 for June from 1960 to 1968?
> What is the standard deviation?
> List all the flows at the main stem of the Tolt for 1972.
> Plot the site 15 flows for May all years.
> At site 7, what is the lowest flow for all the years on record?
> Sum the flows from June to September for 1969?

The processor is not case sensitive (upper-case is not needed) and punctuation at the end of the sentence is not required. Not all the keywords need be present in order for the sentence to be processed. When some keywords are missing, the processor either uses the default values, or assumes that the question was asked in the same context as the previous question. In this later case, keywords from the previous sentence are used for the missing ones.

The Prolog code for the natural language processor consists of three main components: (1) clauses for parsing the sentence, (2) clauses defining the lexicon (dictionary) which contains the words recognized by INLET and their associated synonyms, and (3) clauses used to determine the command for that particular grouping of keywords and the context of the sentence.

The first group contains all the clauses associated with reading the sentence and selecting the words recognized by the system. This is accomplished by reading each word, one by one, and evaluating the clauses contained in the lexicon. If the word is not one contained in INLET's dictionary, it is ignored and the parser moves to the next word. If it is recognized, the parser uses the lexicon to determine the word class to which it belongs. It then evaluates the keyword by examining the synonym clauses in the lexicon. For example, the word "mean" is a synonym for "average." Both of these words are recognized by INLET and belong to the word class "statistic," but "average" is the actual keyword used by the system. If the

parser encounters "mean," it finds that it is a statistic and the associated keyword is "average." "Average" is then passed to the clause that actually executes the command.

Some words, as indicated in Table 4-1, are not single words (i.e. "Site 1"). In this case the actual word recognized by INLET is "site." When the system encounters "site," however, it expects to find a number after "site." INLET then reads the next word to see if it is a number. If so, this number is used to determine the keyword for that particular site.

The third group of clauses translate the list of keywords into the command. One of the rules used is:

> If the *command* is **plot,**
> and the *statistic* is **all,**
> and the *site* is known,
> and the *month1* is known,
> and *month2* is **none,**
> and *year1* and *year2* are both **none,**
> then plot all the flows for *month1* at *site.*

In this example the variables are italicized and the values of the known keywords are in bold. There are many clauses in this group for processing all the types of commands that INLET performs.

Context Dependency

There are also clauses that determine what to do when not all of the keywords are present. For example, INLET does not require the user to specify the site name each time. Once a site has been selected it remains active until another site is mentioned. If a site is not mentioned in the input sentence, INLET assumes that the previous site is the current one. INLET also determines the meaning of the sentence in context with the previous sentence.

To illustrate this process, suppose the following two commands were given:

"Plot the May flows at the North Fork of the Tolt."
"Plot them at the South Fork."

In the second sentence, the user most likely intended for the *May* flows at the South Fork of the Tolt to be plotted. The second sentence is within the context of the previous sentence. The only words recognized by INLET in the second sentence, however, are "plot" and "South." This is an ambiguous command for INLET because no time period or month is specified. In this case, INLET refers back to the keywords used in the previous sentence to find the right keywords to use, i.e. "may."

Context determination is made possible by storing the keywords from the previous sentence in Turbo Prolog's internal database. An internal database is a collection of clauses that can be added to, or retracted while a program is running. INLET creates an internal database containing a "context" clause. If insufficient keywords are present for INLET to determine the meaning of the sentence, the context clauses are evaluated to determine the values of the missing keyword(s). After all required keywords are assigned values, the previous context values are retracted and the new context is asserted into the database. For example, if a keyword for site name is not found in the current sentence, the program searches the context clause to find the keyword from the previous sentence and then asserts the site name to be equal to the previous one.

Context dependency is an essential feature of INLET that is often not available in simple natural language processors. This feature is extremely valuable for increasing the user friendliness of the program and decreasing the number of words required to express a command.

Help Features

Help generally will not be needed. There are some situations, however, in which the user may need assistance. The user can type "help" at any time to see a

list of the keyword groups and the words recognized by INLET in these groups. This command also lists the site names.

The first question in an INLET session must specify a site name in order for INLET to execute the command. If none is present, a menu listing the sites appears and the user may choose the appropriate site. The menu can also be called by entering "site" or "sites," without any site number following. The menu can serve as a reminder of the sites available and their site number and common names. Again, once a site is chosen it will be used until a new one is specified in the command or chosen from this menu.

Sometimes insufficient keywords are found because of misspellings or because the input sentence used words not recognized by the lexicon. In this case, INLET informs the user that the sentence cannot be processed. If the year is out of range, or a site name is used that does not correspond to one of the available sites, INLET will also indicate that and prompt the user to input a new year or site name.

INLET also has a menu-driven system which provides a review of the options available for queries. This menu system is accessed from the natural language processor by entering **<ESC>**. The menu system offers the same statistical and plotting capabilities as the natural language processor. The optimal reservoir operation data, however, are not available through the menu system. These operational questions are more complicated and would require a large number of menus. They are easily accessible from the natural language processor, however.

SUMMARY

INLET is an interactive environment designed to provide easy access to streamflow and operation policy data. Access to this data can be gained either through a menu system or through a natural language interface. These interfaces

have been constructed in the Prolog language and have been made executable on personal computers compatible with the IBM AT. INLET provides the user with a wide range of statistical options and plotting capabilities.

The following chapter illustrates the use and application of INLET to a specific problem.

CHAPTER 5

FEATURES AND APPLICATION OF INLET

This chapter discusses how INLET is used to provide convenient access to information necessary for proper water resource planning and management. The menu system and natural language system are described and the capabilities of the system are outlined. Typical interactions with INLET are presented as well as figures of graphical output. It is unnecessary for a user to refer to any manual or this chapter to use INLET. When INLET is activated, the user initially is provided a menu listing three options. Those options allow access to the natural language system, the menu system, or to DOS.

INLET FORMAT

Although the features available with either the menu system or the natural language system are similar, their use is very different. Menu-driven systems are familiar to any user of advanced computer software. Menu-driven systems allow a user to choose one or more items from a list of items. The selection of the items typically is accomplished either with a mouse or by moving a cursor. When commands are hierarchical, (that is, several steps are required), numerous menus must be evaluated to complete a command.

For instance, suppose one wishes to plot all of the data at a particular streamflow gauging site. The user must indicate this desire to the computer through a series of menus. The user first indicates the site location, next the fact a plot is desired, next, the time period for which the data should be included, and finally, the data type (all data, or just data for one month, etc.). In this process the user is required to select from four menus.

A natural language interface provides the user with a completely different approach to this process. The user types the command such as, "Plot the June flows at Site 1 from 1950 to 1960." All of the information needed is contained in the single command. Unlike the menu approach, the user gives the command to the computer just as he might naturally state the command.

As indicated by the literature reviewed in Chapter 2, natural language interfaces have gained significant popularity for accessing databases. The primary advantages of a NLI over a menu system are (Hayes, 1986):

(1) Natural language interfaces are typically more versatile; they can answer a wider range of questions.

(2) Natural language questions are more direct. To access the same data with the menu system would require selecting many menus.

(3) In many complex situations, natural language systems take less time for the user.

(4) Natural language systems allow users to formulate questions in a manner that is consistent with the way they think about the problem.

In the following sections, INLET's use of both a menu system and a natural language interface is described. An example session using INLET to analyze the potential operating strategies for a water supply system is presented.

MENU SYSTEM

The menu system provides access to data retrieval, statistics and plotting functions. The menu system is accessed by choosing the second item on the main menu. (To return to a previous menu at any time, the user can press the <ESC> key.) A status line, located at the bottom of the screen, informs the user how to advance or return in the menu system. Selection of an item from a menu can be performed in two fashions. To indicate the choice of an item from the menu, the user either moves

the highlighted bar to his choice and presses the **<ENTER>** key, or types the first letter of the line of their choice.

After choosing the menu system selection, the second menu appears containing four choices: Choose Site, Statistics, Plotting and Data retrieval. These choices are described under the next four headings.

Choose Site

Before plotting, statistical analysis, or data retrieval can be performed, a site must be selected. For this application, the seven primary sites used by the Seattle Water Department for water management planning are incorporated into the system. It should be noted that any number of sites could be placed in INLET. There is nothing unique about the sites chosen for this case study. The site selection option has a menu listing the seven sites available and their associated common names (those used by the Seattle Water Department planning staff). To select a site, as with any other menu selection, the user positions the cursor on his selection and presses **<ENTER>**. When a site is selected, all other statistical and plotting commands are assumed to reference that site. In other words, a site remains 'active' until the user selects another site.

Statistics

Statistical analysis is important for evaluating streamflow data. It is important to place a statistical context on data so that reliable comparisons can be made. The statistics available in INLET are mean, standard deviation and skew. These statistics can be performed for one month, all years; for one year, all months; or for one month between two specified years. When the statistics option is selected, another menu appears with a list of the statistics. After choosing a statistic, another menu appears containing the choices: month, year and specify time period.

If "month" is chosen, another window will request the user to enter the month desired. The statistic will then be calculated for that month for all the years on record. If "year" had been chosen, a window would allow the user to input the desired year and the statistic would be calculated using all monthly data for that year. Specifying the time period creates a series of windows requiring the user to enter the beginning and end years, and whether all monthly data is desired or just one month. The calculated statistics are displayed in a window in the upper half of the screen.

Plotting

Plotting streamflow data is a very effective procedure for evaluating current conditions relative to previous events. For example, a plot of the historic flows ranked from lowest to highest (the cumulative distribution function) readily indicates how many years on record a particular month had streamflows above (or below) a specific value. A plot of all the flows on record for a particular month between two specified years (a time series plot) indicates the natural variability of flows over time.

When the plotting option is chosen, another menu lists the plotting options: Time Series or Cumulative Distribution Function. Choosing either one of these options creates a window with the options: "Single Month" or "All Months." If the "Single Month" option is selected, another menu appears requesting the user to enter the month and to indicate whether the plot is to be made of all streamflows on record or between two specified years. The "All Months" option generates a plot of all the streamflows on record or all the flows between two specified years. Choosing this option creates a series of menus similar to the "Single Month" option in which the time period is specified. When all the options are selected, the screen is cleared and

a plot is generated covering the screen. The user can press any key to return to the menu system.

Data Retrieval

The fourth option from the second menu, "Data Retrieval," is used to retrieve monthly streamflow values from any of the sites. The ways in which the data can be retrieved are a list of streamflows for one month, a list of all the flows for one year, or all the flows between a specified yearly interval. These options are titled: "Single Month," "One Year," and "All Months." Selecting the "Single Month" and "All Months" generates a series of menus identical to those for specifying the month and time period for the plotting options. When "One Year" is selected, the user is prompted to enter the year number. The streamflows are displayed in a window covering the upper half of the screen.

NATURAL LANGUAGE SYSTEM

After choosing the natural language interface (NLI) from the main menu, the computer screen is cleared and two windows appear, covering the upper and lower half of the screen. The lower window, which has a prompt and a cursor serves as the query window. When the user types a question, it is displayed in this lower window. INLET's response to the question appears in the upper window. If the command is to plot data, the screen is cleared and a plot is displayed covering the entire screen. A status line at the bottom indicates how to return to the query window (hit any key to continue). To return to the main window from the query window the user presses the <ESC> key.

A status line at the bottom of the screen reminds the user how to return to the menu system and how to recall the previous command for editing (<F8>). Pressing the <F8> key displays the previously entered command with the cursor at the

beginning of the line. The editing keys can then be used to change the input. At any time the user can enter "Help" to receive instructions and guidance with the natural language system.

Features Unique to the NLI

Although a majority of the features of INLET are available from both the menu system and the natural language interface, one important feature is only available in the natural language interface. As indicated previously, INLET is developed to aid water managers in evaluating the conditions of a water supply system and developing operational strategies. The large operational database of the SWD has been incorporated into INLET through the natural language interface.

This database indicates the optimal operating policy for any year on record, given an initial storage level, demand level and beginning month. The information indicates situations in which water-use restrictions may be necessary to minimize the impacts of low-flow conditions. The use of this feature of INLET is explained in detail in the following section.

TYPICAL INLET SESSION

Consider a situation in which a water manager finds himself in a particular month, with low streamflows, and the reservoir system at thirty percent of its capacity. The manager is interested in a variety of information including:

1) How unusual are the current streamflows?
2) How likely are the low flows to continue?
3) What is the probability of flows being as low as the ones currently experienced?
4) Should water use restrictions be initiated?
5) When should they be initiated and how stringent should the restrictions be?

It is possible that a wide range of other questions may occur to the manager. It is also likely that the order in which the manager wishes questions to be answered

will vary from one session to another. The setting that is suggested here is a common one. A somewhat unusual event (low streamflows) has occurred and the manager wishes to place this situation into a context that will help him develop a reasonable response plan.

The purpose of INLET is to allow the manager to pose these questions in any order and to analyze the situation to the extent necessary. INLET accomplishes this by providing the wide range of statistical analysis tools previously described, access to the operational database, and total flexibility in the extent and order in which the analysis is made.

The following example is offered to illustrate how one might evaluate a particular low-flow situation. It is important to realize that the questions could be posed in any order or that entirely different questions could be raised. INLET allows the user to analyze a situation in any manner he wishes.

After reviewing recent streamflows and the current storage levels of his system, the water manager's first step is to find the mean and standard deviation of the June flows. He decides to use the menu system first. The first menu to appear is the main menu shown in Figure 5-1. From this menu, the user chooses the menu-driven system. Next, a menu appears containing the statistics, plotting and site selection choices. From this menu, he chooses site selection and the site menu appears. These two menus are shown in Figure 5-2. After choosing a site, the site menu automatically disappears and the user is returned to the second menu from which he selects "Statistics." The statistics menu appears providing statistical options: mean, standard deviation and skew. After choosing mean, a menu appears with the time period options: month, year, or specify time period. He chooses each option in turn to find the mean of all the June streamflows on record, the mean of

1972, and the June mean from 1950 to 1960. The answers to these selections are displayed in a window covering the top half of the screen. Figure 5-3 shows the menus associated with these selections and the results. To find the standard deviation, the user would enter <ESC> until he is back at the statistics window. From there he would choose "Standard Deviation" and repeat a similar process to that of specifying the time periods when he determined the various means.

Having evaluated the basic statistics, the user now wants to view two plots, a time series plot of the June flows and the cumulative distribution function of those flows. He returns to the second menu (the menu on the left in Figure 5-2). From that menu, he chooses "Plotting." A system of menus appears from which he selects the time period to plot (Figure 5-4). He specifies that he wants a time series plot of monthly streamflow data for all years on record for the month of June. The resultant plot is shown in Figure 5-5. From this figure the manager recognizes immediately the high variability of the June streamflows.

Next, the manager wants to view a cumulative distribution function (CDF) of the June streamflows. He presses the escape key until the menu with "Cumulative Distribution Function" appears. After choosing this option, a similar set of menus appears to that of the time series plot, and he specifies the month and desired time period. From the CDF (Figure 5-6), the user can compare the current streamflow to a ranking of all the streamflows on record.

By pressing any key, the user returns to the second menu. He decides to plot all the streamflows for the year 1970. He chooses the "Plotting" option from the second menu and year from the next menu that appears. He then specifies "1970" and hits return. Figure 5-7 shows the plot that is created.

The user now decides that he wants to use the natural language system and ask some similar questions for comparison with the menu-driven system. He enters the escape key until he is at the main menu (Figure 5-1) where he chooses the first option, "Natural Language Interface."

In the query window the user enters his questions about the mean and standard deviation for the June flows. Figure 5-8 illustrates a series of questions posed using the natural language interface and INLET's responses. It should be noted that all of the questions are posed in simple English statements. The responses by INLET provide a complete reply and answer to the questions. The last request is "Please plot all the streamflows at Site 16 between 1950 and 1960," which creates the plot shown in Figure 5-9.

After carefully reviewing all of the flow data, the user decides he now wants to analyze some reservoir operation policies. From his previous analysis of streamflow data, he determines that the current flow conditions are similar to those experienced in four different historic sequences, 1930, 1938, 1954, and 1959. To investigate the optimal operating pattern associated with those time periods, the user wishes to review the appropriate data. To do so, the user enters "Please show the operating policies for June, 1930, with initial storage of 30%." The results of this command are given in Figure 5-10. After reviewing the restriction policies for 1930, the user can review those of 1939, 1954, and 1959.

SUMMARY

This chapter presented an introduction to the use of INLET and describes its application. Such a program allows someone unfamiliar with computers but knowledgeable about a problem area to thoroughly analyze a difficult problem

without extensive computer training. The value of such tools, and the manner in which the program can be improved, is the subject of the final chapter.

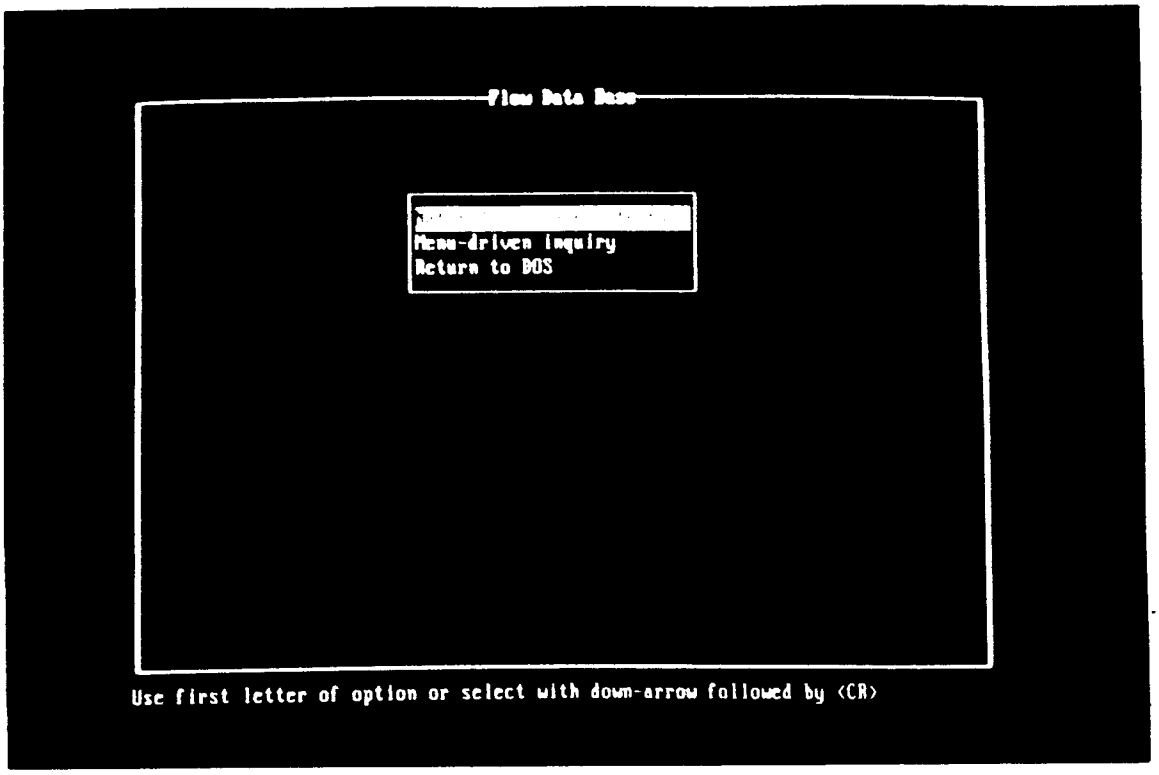


Figure 5-1. The Main Menu.

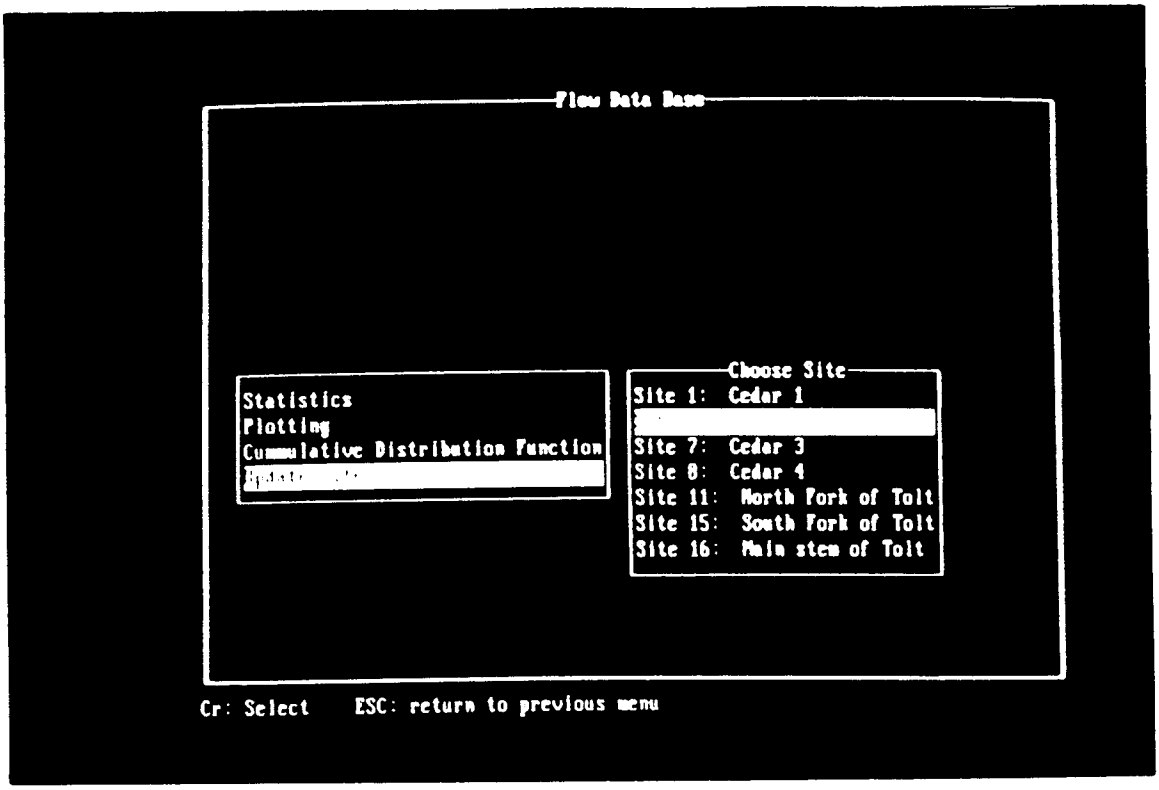


Figure 5-2. Site Selection.

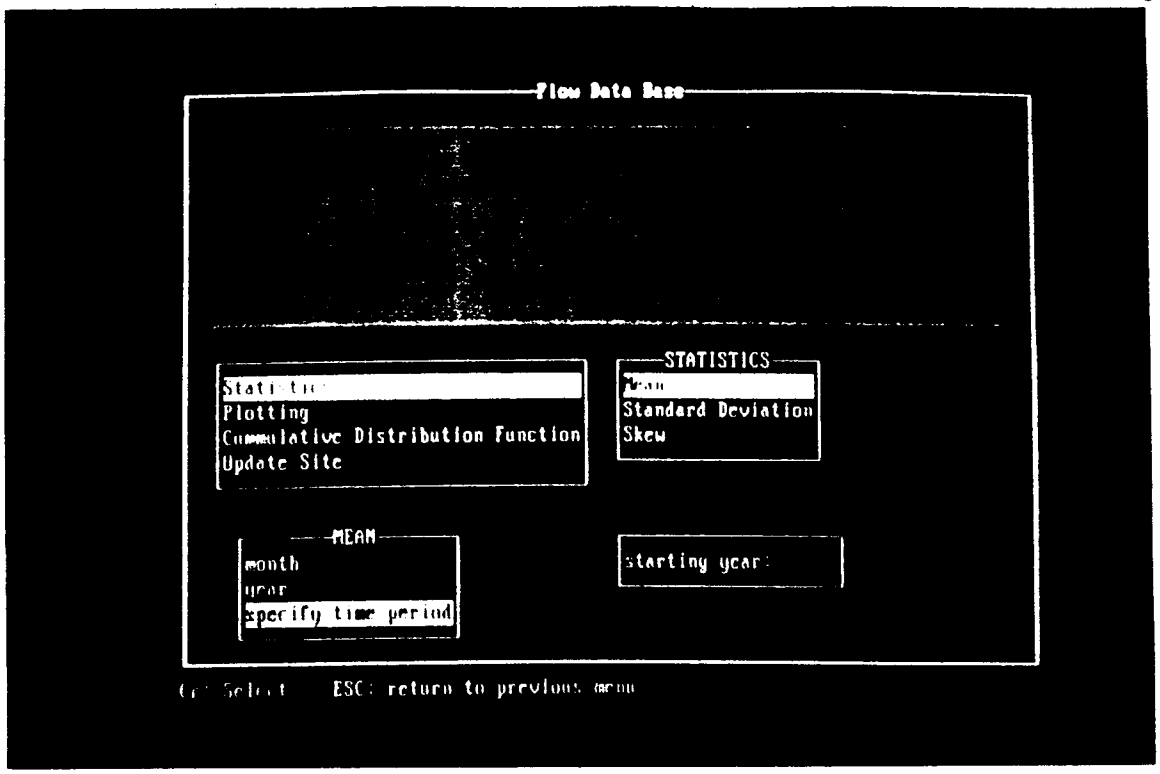


Figure 5-3. Determining Statistics Using the Menu System.

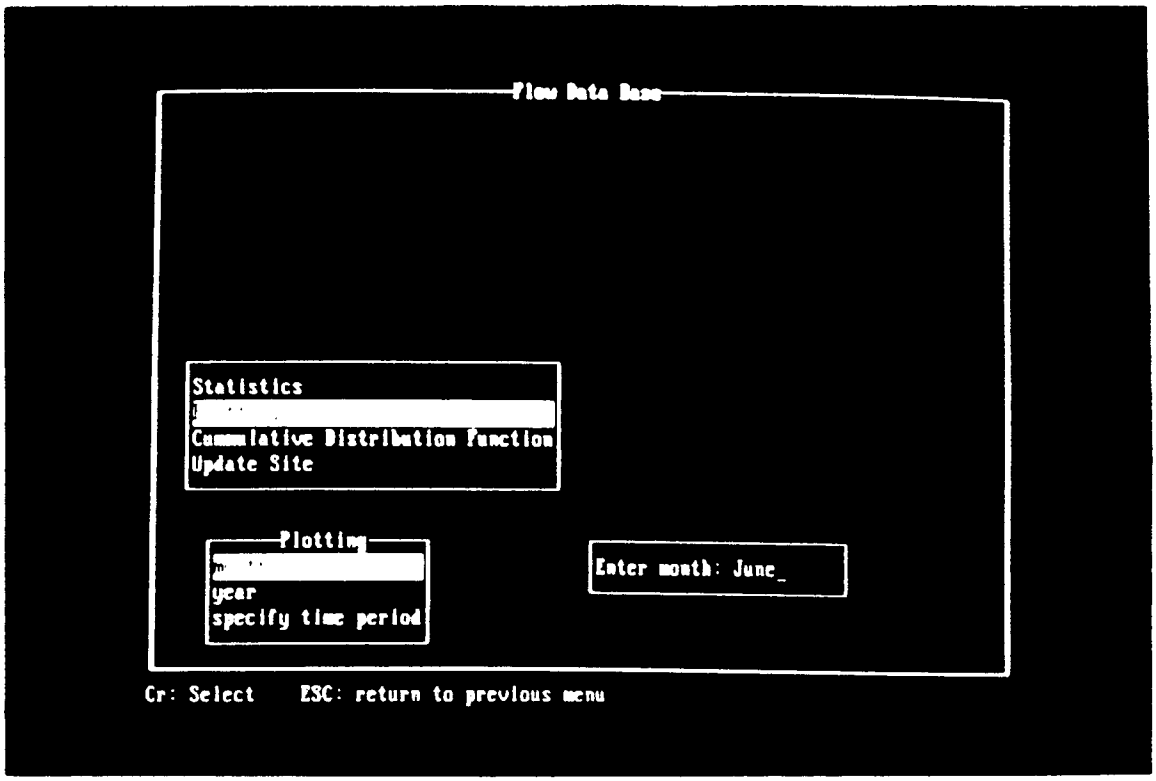


Figure 5-4. Menus Specifying Plotting Options.

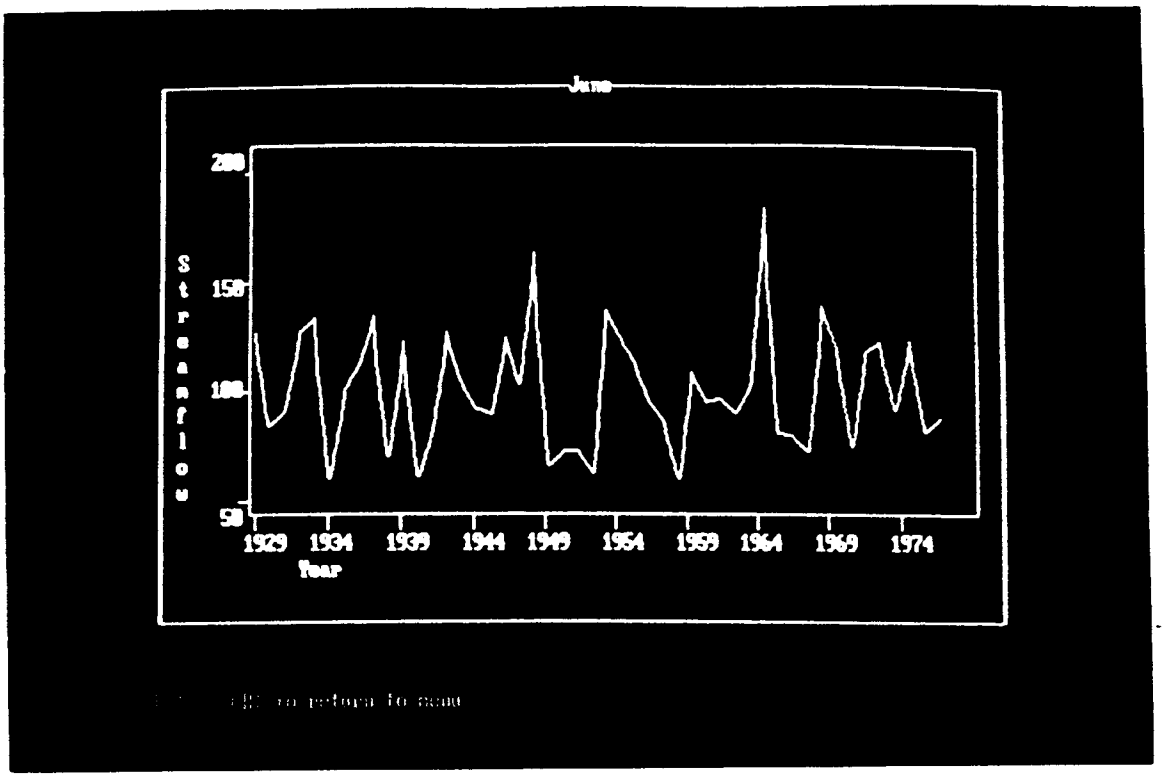


Figure 5-5. Time Series Plot of all the June Streamflows on Record.

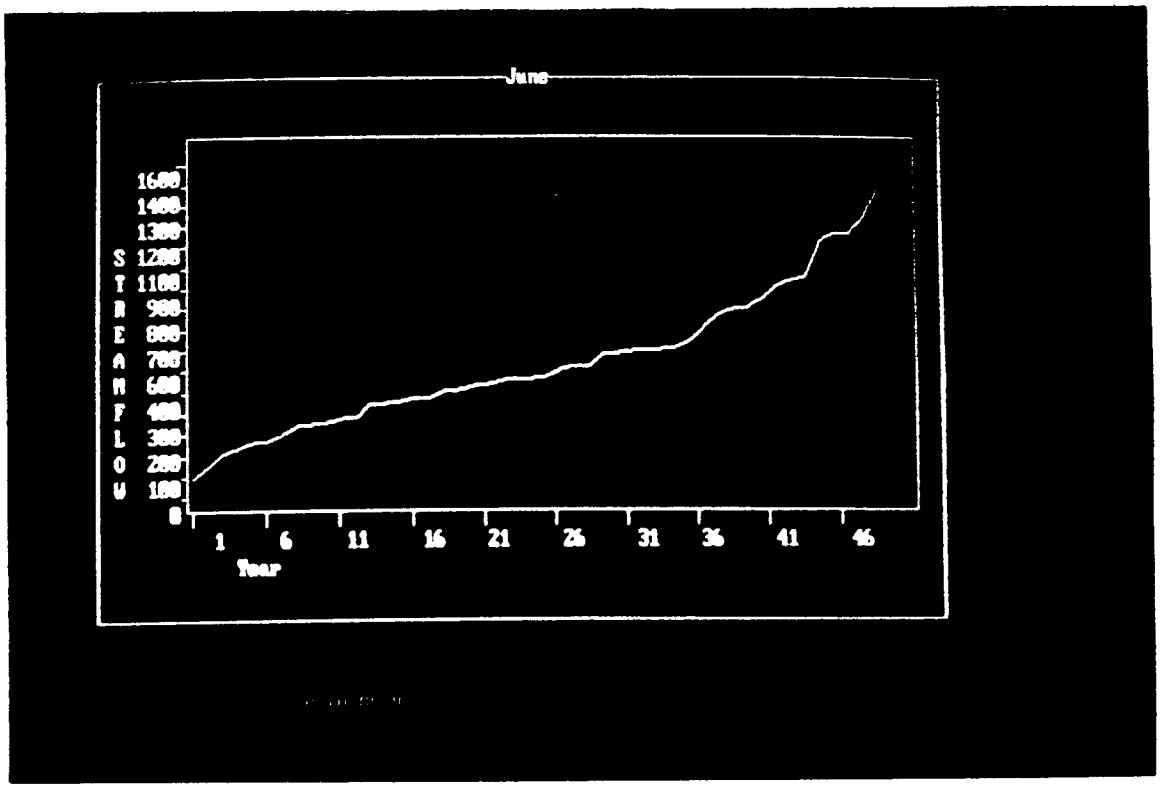


Figure 5-6. CDF of the June Flows for all Months on Record.

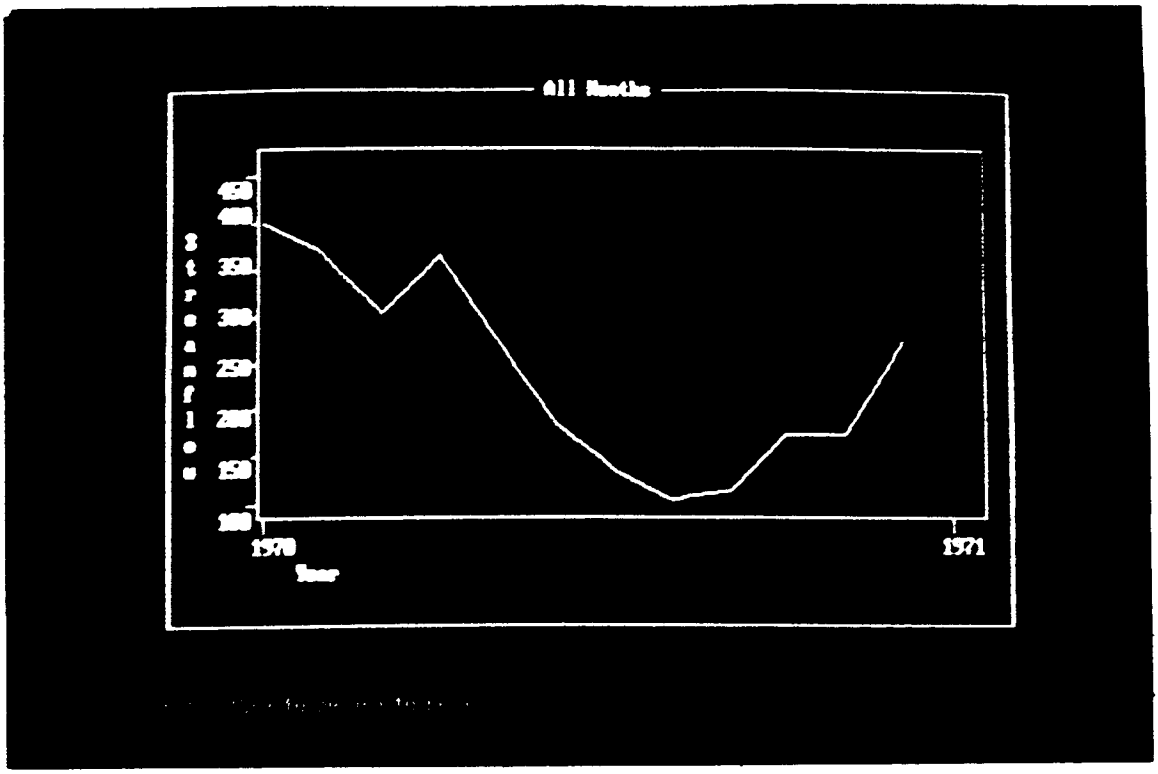


Figure 5-7. Plot of all Streamflows in 1970.

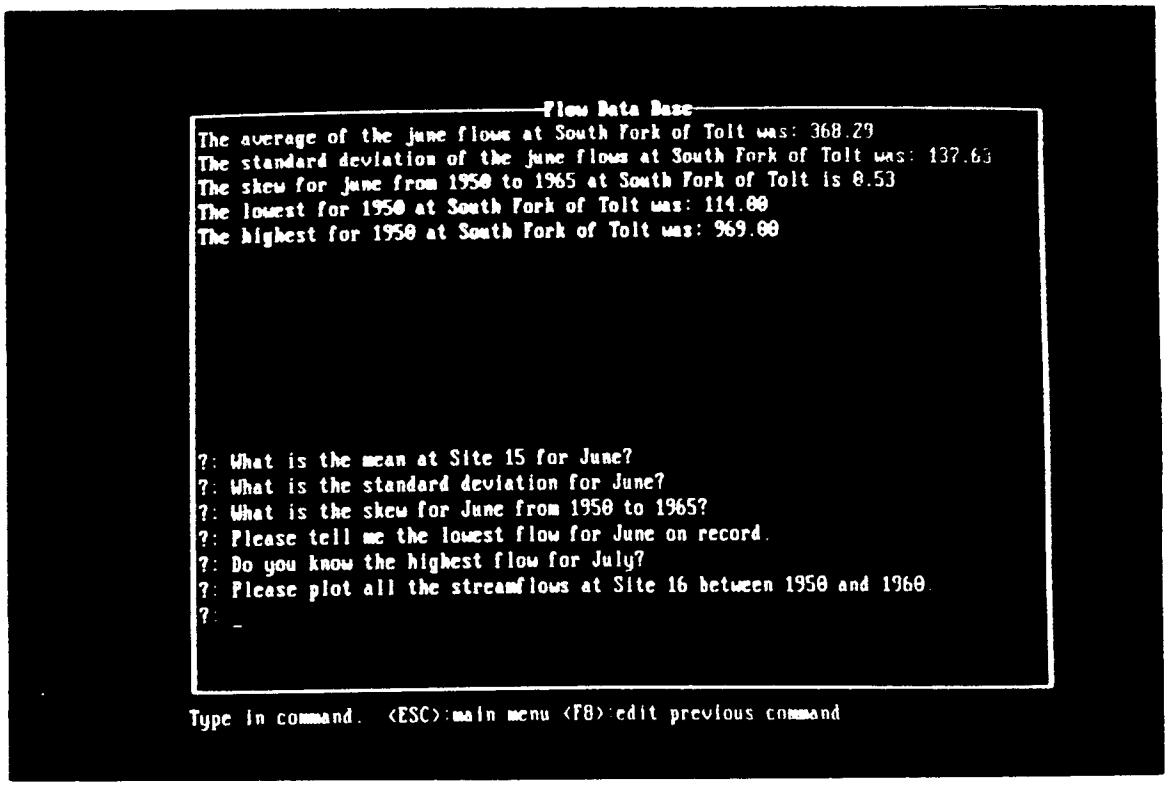


Figure 5-8. The Natural Language Interface.

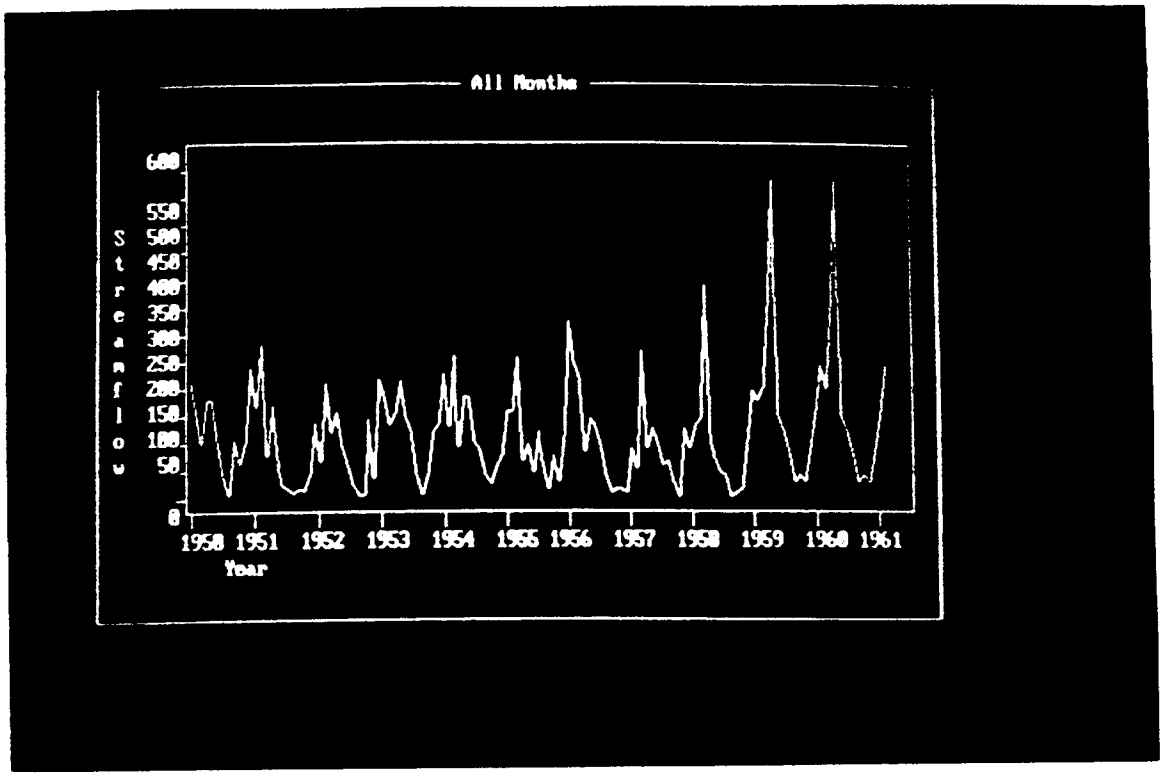


Figure 5-9. Plot of all the Streamflows at Site 16 Between 1950 and 1960.

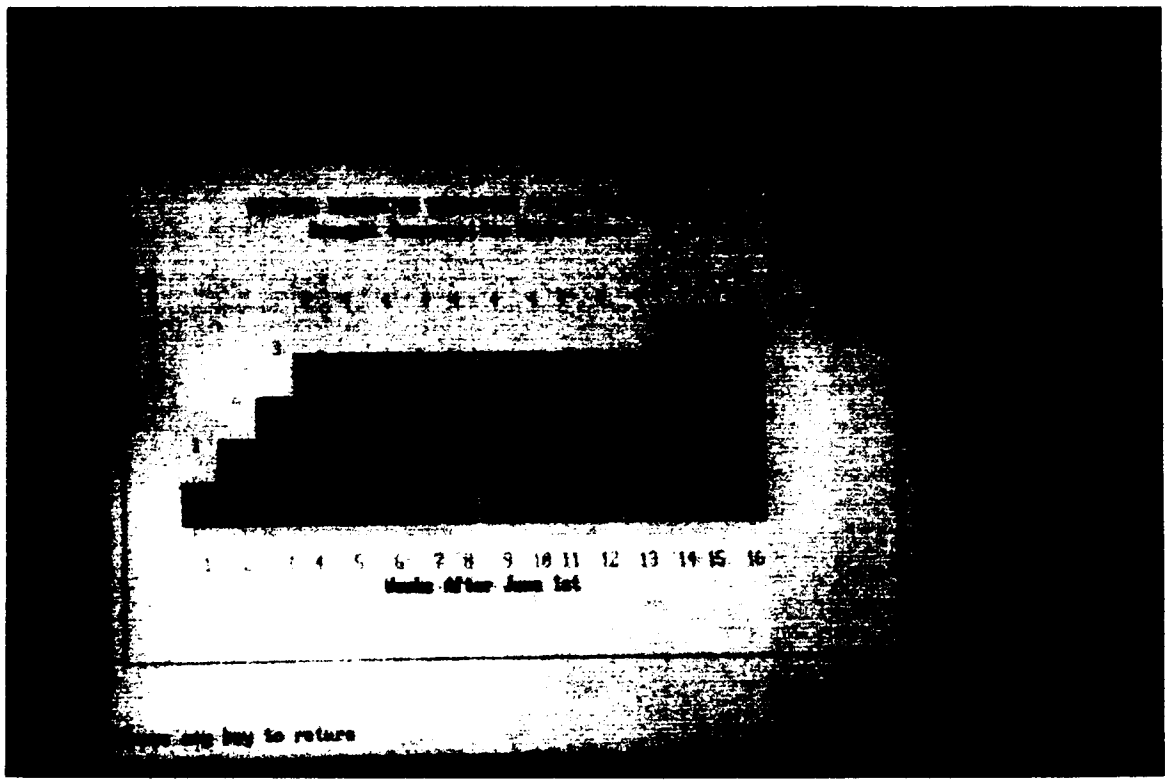


Figure 5-10. Optimal Reservoir Staging Levels.

CHAPTER 6

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

Summary

This paper discusses the need for user-friendly and flexible computer model interfaces in water resources planning and management to facilitate the use of complex computer models by water managers. Natural language processing has been presented as means to meet this need. This approach allows managers to make effective use of computer models with a minimum of computer expertise or training. This is accomplished by developing interfaces to the models that are natural, easy to use, and controlled by simple statements in standard English (i.e.: "Please plot the streamflows for the Cedar River from 1929 - 1976"). The development of such interfaces allow the managers to concentrate on evaluating and interpreting the output of a model rather than on learning a new set of computer commands. The interfaces allow the manager an increased level of communication with the models and therefore he can direct the model development process more effectively.

This paper discusses an approach taken in developing such an interface, an computer program called INLET (for Interactive Natural Language Environment). This program provides a wide range of graphical and statistical functions for evaluating streamflows and drought management data. Data for the Seattle Water Department has been incorporated into the model to display its features.

Chapter 1 provides an overview of the report. In this chapter the changing role of water resource managers and water resource analysts is described. These changes have resulted from the introduction of microcomputers into the analysis process. The types of models and analysis tools now available to aid water resource decision makers has altered dramatically the way in which computers can be

incorporated into the decision making process and their potential value in improving the quality of decisions made.

Chapter 2 provides a literature review of water resources planning models and a discussion of natural language interfaces. This chapter suggests that as models and analysis techniques become more user-friendly, decision makers will be able to use complex models without the aid of other analysts. This will allow the decision makers to make more use of such models and increase the effectiveness of the decision making process. Natural language processing is shown to be valuable as a user-friendly interface to a database. This is very important in water resources planning and management, as it makes data directly available to decision makers who may not have time to learn a formal database query language or wait for results from others.

In Chapter 3, the recent investigations of the Seattle Water Department (SWD) are described together with a drought management expert system. A detailed discussion of the components of the SWD system is given with a brief history of the evolution of models that are used in planning and operation. A large operational database that was developed using an optimization model is also described. This database is used in later stages of this research.

The development of INLET is described in Chapter 4. INLET is an interactive natural language environment providing access to water resources data. INLET was written in the Prolog programming language which is a symbolic computer language well suited for developing user-friendly environments, particularly natural language interfaces. Turbo Prolog has many extra features such as graphical and menu functions used by INLET.

INLET has two different methods for accessing the data, plotting and statistical functions. These two components are a menu-driven system and a natural language interface. The natural language interface uses a noise-disposal parser to find the keywords in the natural language command. This approach is well suited for a database interface because the domain is limited and database queries can easily be decomposed into a series of keywords. Noise-disposal parsers allow the user to form questions in any manner as long as the keywords are present. INLET also accepts questions which are context-dependent, for example, questions which refer to previous questions. This feature decreases the number of words that need to be typed and accepts more naturally formed questions.

Chapter 5 describes the data retrieval abilities and statistical and plotting functions of INLET. These functions are available from both the natural language interface and the menu-driven system. Optimal reservoir operation strategies are available through the natural language interface. This operation data was generated by an optimization model and forms the database used in the drought management expert system. One of INLET's strengths is that it can be used in conjunction with the drought management expert system to analyze reservoir operation policies. These features are described and a sample session demonstrating their use is presented.

Conclusions

1) The evolution toward more user-friendly software will allow managers to participate more effectively in model development and use. This will increase communication between model developers and managers. With increased communication, models can be developed that will more directly serve their users.

2) Prolog has demonstrated itself to be an ideal language to develop user-friendly software. Using Prolog makes natural language processing possible in an efficient and fast manner. This suggests that more engineers should become familiar with non-procedural programming languages.
3) INLET allows novice computer users access to complex data and provides very useful statistical and plotting capabilities. Although INLET is as yet untested on water resource managers at the Seattle Water Department, it has been demonstrated at several conferences and the feedback from its users is very positive. This ability to provide direct data access to novice computer users has significant implications on the use of such models by water resource managers. These managers will now be able to explore a large number of operational policies and their impacts on system reliability with ease.
4) Interaction with SWD personnel suggest that such software can be incorporated into their planning process. The data, previously difficult to access in a timely manner, are now directly available to all SWD staff and can be used to improve their decision making.

Recommendations

1) Interact with the SWD to incorporate INLET into their planning process

   The initial reaction of staff at SWD is positive. The next step is to test INLET and then tailor it more closely to their planning process. Specific needs can be identified and then addressed by INLET.

2) Develop additional statistical and plotting functions

   Based on the results of testing INLET with SWD staff, desired additional statistical functions would be identified. For example, determining the correlation between two sites, might be identified as being valuable in a

decision analysis process. Plotting functions could also be expanded to allow data from two sites to be plotted simultaneously. More general questions such as, "What years on record are similar to the current situation?" may be useful to compare optimal reservoir operation policies from years similar to the current year.

3) Analyze user preferences

Define the conditions under which a user prefers the use of menu systems or natural language interfaces. These conditions could be best defined after extensive testing. User testing, for example, may show that the best approach is a more-closely integrated approach in which a menu-system is used for many simple requests and a natural language window, appearing on the same screen, could be used as needed.

4) Investigate the use of INLET with other large databases

The technology allowing the transfer of this interface to other water resources databases should be developed. Ideally, facilities should exist to incorporate streamflow data for any stream in the state of Washington.

BIBLIOGRAPHY

Bellman, R.E., An Introduction to Artificial Intelligence: Can Computers Think?, Boyd and Fraser Publishing Company, San Francisco,1978.

Bellman, R.E., and Dreyfus S.E., Applied Dynamic Programming, Princeton University Press, 1962.

Boose, J.H., Expertise Transfer for Expert Systems Design, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1986.

Borland International, Turbo Graphix Toolbox, Scotts Valley, CA, 1985.

Borland International, Turbo Prolog: Reference Guide, Version 2.0, Scotts Valley, CA. 1988.

Brachman, R.J. et al., "What are Expert Systems?," In Building Expert Systems, ed. Hayes-Roth F., Waterman, W.A., and Lenat D.B., Addison-Wesley Publishing Co. Inc., Reading, MA, 1983.

Buchanan, B.G., and Shortliffe, E.H., ed. Rule Based Expert Systems, Addison-Wesley Publishing Co. Inc., Reading, MA, 1984.

Duda, R., Gaschnig, J., and Hart, P., "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," In Expert Systems in the Microelectronic Age, ed. Michie, D., Edinburgh University Press, 1979.

Earley, J., "An Efficient Context-Free Parsing Algorithm," Communications of the ACM, 13, February, 1970.

Fedra, K., and Loucks, D.P., Interactive Computer Technology for Planning and Policy Modeling," Water Resources Research, Vol.21, No.2, February, 1985, pp 114-122.

Green, B., Wolf, A., Chomsky, C., and Laughery K., "BASEBALL: An Automatic Question Answerer," Proceedings of the Western Joint Computer Conference 19, 1961, pp 219-224.

Grishman, R., Computational Linguistics, Press Syndicate of the University of Cambridge, New York, NY, 1986.

Harmon, P., and King, D., Expert Systems in Business, John Wiley & Sons Inc., New York, NY, 1985.

Hayes-Roth, F., Waterman, W.A., and Lenat, D.B., "An Overview of Expert Systems," In Building Expert Systems, ed. Hayes-Roth, F., Waterman, W.A., and Lenat, D.B., Addison-Wesley Publishing Co. Inc., Reading, MA, 1983.

Hendler, J. and Lewis, C., "Introduction: Designing Interfaces for Expert Systems," in Expert Systems: Designing the User Interface, ed. Hendler, J., Ablex Publishing, Norwood, NJ, 1988, pp 1-13.

Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., and Slocum J., "Developing a Natural Language Interface to Complex Data," ACM Transactions on Database Systems, Vol.3, No.2, June 1978, pp 105-147.

Hendrix, G.G.and Walter B.A., "The Intelligent Assistant," BYTE, December 1987, pp 251-258.

Hirsch, R.M., "Synthetic Hydrology and Water Supply Reliability," Water Resources Research, 15(6), 1979, pp1603-1615.

Kunreuther, H., and Miller, L., "Interactive Computer Modeling for Policy Analysis: the Flood Hazard Problem," Water Resources Research, Vol.21, No.2, February, 1985, pp 105-113.

Lane, A., "DOS in English," BYTE, December, 1987, pp 261-263.

Loucks, D.P., Kindler, J.and Fedra, K., "Interactive Water Resources Modeling and Model Use: An Overview," Water Resources Research, Vol.21, No.2, February, 1985, pp 95-102.

McDermott, J., "R1: The formative Years," AI, 2(2), 1981.

Martin, M., ed., Expert Systems 85: Proceedings of the Fifth Technical Conference of the British Computer Society," Cambridge University Press, Cambridge, U.K., 1985.

Obermeier, K.K., "Natural-Language Processing," BYTE, December 1987, pp 225-233.

Palmer, R.N., and Holmes, K.J., "Operational Guidance During Droughts: Expert System Approach," Journal of Water Resources Planning and Management, Vol.114, No.6, November, 1988.

Palmer, R.N., and Johnston, D.M., "Completion Report for Optimization of Yield Analysis on Seattle Water Supply System," Report, Seattle Water Department, Seattle, WA, 1984.

Palmer, R.N.and Tull, R.M, "Expert System for Drought Management Planning," Journal of Computing in Civil Engineering, Vol.1, No.4, October 1987.

Politakis, P.G., Empirical Analysis for Expert Systems, Pitman Publishing Inc., Marshfield, MA, 1985.

Quade, E.S.,and Miser, H.J., Handbook of Systems Analyses, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1983.

Reboh, R., Reiter, J., and Gaschnig, J., Development of a Knowledge-Based Interface to a Hydrological Simulation Program, SRI International, Menlo Park, CA, 1982.

Schildt, H., Advanced Turbo Prolog, McGraw-Hill, Inc., Berkeley, CA, 1987.

Seattle Water Department, Seattle Comprehensive Regional Water Plan, 1985 COMPLAN, vol. 6, Seattle Water Department, Seattle, WA, 1986.

Shortliffe, E.H., Axline, S.G., Buchanan, B.G., Merigan, T.C., and Cohen, S.N., "An Artificial Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy," Computers and Biomedical Research, Vol. 6, 1973, pp 544-560.

Simmons, R.F., "Man-Machine Interfaces: Can They Guess What You Want?," IEEE Expert, Spring, 1986, pp 86-94.

Sneiderman, B., Designing the User Interface: Strategies For Effective Human-Computer Interaction, Addison-Wesley, Reading, MA, 1986.

Sunset Software Technology, 1987, XA Profession Linear Programming System, San Marino, CA.

Templeton, M., and Burger J.,"Considerations for the Development of Natural-Language Interfaces to Database Management Systems," in Cooperative Interfaces to Information Systems, ed. Bolc, L., and Jarke, M., Springer-Verlag, Berlin, Germany, 1986, pp 67-99.

Townsend, C., Advanced Techniques in Turbo Prolog, Sybex Inc., San Francisco, CA, 1987.

URS Corporation, Water Resources Management Simulation Model Programmers Manual. URS Corporation, Seattle, WA, 1981.

Weizenbaum, J., "ELIZA -- A Computer Program for the Study of Natural Language Communication Between Man and Machine," Communications of the ACM, 9, January, 1966.

Winston, P.H., and Horn, K.P, LISP, Addison-Wesley, Reading, MA, 1981.

Woods, W.A., "Transition Network Grammars for Natural Language Analysis," Communications of the ACM 13, 10 (October), 1970, pp 591-606.

Woods, W.A., "Progress in Natural Language Understanding: An Application to Lunar Geology," AFIPS Conference Proceedings 42, 1973, pp 441-450.

Woods, W.A., "Semantics and Quantification in Natural Language Question Answering," Advances in Computers, Vol 17, Yovits, M., Ed., New York: Academic Press, 1978, pp 2-64.

APPENDIX: THE INLET CODE

```
/*------------------------- NLP_POL.PRO ---------------------------------*/
%**** The driver module and language clauses

code=3800
%project "NLP"

include "nlp_inc.pro"
database - current_context                    %--- stores current context of request
   context(symbol,symbol)
   old_context(symbol,symbol)
   current_color(integer)
include "nlp_db.pro"
include "nlp_stat.pro"
include "nlp_flow.pro"
include "nlp_plot.pro"

predicates
/*---------- Parsing and Language ----------*/
          start_language
          description(string,string)
          next_word(sentence,sentence,symbol)
          find_delim(sentence,integer,integer)
          strip_space(sentence,sentence)
          process(sentence)
          escape(sentence)
          next_valid_word(sentence,sentence,symbol)
          word_or_number(symbol)
          get_command(sentence,sentence,symbol)
          get_demand(sentence,sentence,symbol)
          get_demand_number(sentence,sentence,symbol)
          get_site(sentence,sentence,symbol)
          get_sitenumber(sentence,sentence,symbol)
          get_stat(sentence,sentence,symbol)
          get_percent(sentence,sentence,symbol)
          get_month(sentence,sentence,symbol)
          get_year1(sentence,sentence,symbol)
          get_year2(sentence,sentence,symbol)
          get_noise(sentence,sentence,symbol)
          terminator(sentence)
          perform(symbol,symbol,symbol,symbol,symbol,symbol)
          pushwords(sentence,symbol,symbol,symbol,symbol,symbol,symbol)
          assert_context(symbol,symbol,symbol,symbol,symbol,symbol)
          determine_context(symbol,symbol,symbol,symbol,symbol,symbol)
          establish_new_context(symbol,symbol,symbol,symbol,symbol,symbol)
          read_context(symbol,symbol,symbol,symbol,symbol,symbol)
          establish_old_context
          all_or_none(symbol)
          show_memory
          purge
```

```
        symbol_string(symbol,string)
        units(symbol,symbol)
        list_commands(string)
        write_list(list)
        write_list(flowlist)
        getpolicy(string,string,string,string)
        run_policy(string,string,string)
        read_policy(flowlist,flowlist,ilists)
        read_years(ilists,ilists,file)
        scan_lines(integer,flowlist)
        scan_lines_aux(integer,flowlist,integer,flowlist)
        scanner(string,flowlist)
        scanner(string,integerlist)
        scanner_int(string,integerlist)
        policy_command(string)

include "nlp_menu.pro"

goal
  flow_data_win_1,
  cursor(10,25),write("Reading in the database"),
  cursor(15,3),
  consultWords,
  consultFiles,
  welcome.

clauses

  start_language:-                          %--- called from nlp_menu
        write("?: "),                       %    starts the parser
        readln(S),!,
        upper_lower(S,Slower),
        process(Slower),start_language.
  start_language:- removewindow,removewindow,
        choose.

  escape(S):- S<>"/".
  show_memory:-                             %--- determines current sizes
        storage(StackSize, HeapSize, TrailSize),  %    of the stack, heap and trail
        write("Stack: ",StackSize),nl,
        write("Heap: ",HeapSize),nl,
        write("Trail: ",TrailSize),nl.

  write_list([]):-!.
  write_list([Head|Tail]):-
        write(Head," "),write_list(Tail).

/*------------------- Language Clauses ----------------------------------*/
  sentence_str(X,Y):- X=Y.                  %--- converts from sentence
  symbol_string(X,Y):- X=Y.                 %    to string, etc.
```

```
description(Site,Desc):-sitename(Site,Name),        %--- makes the site description
        sitenumber(Number,Site),                    %   phrases for answering
        str_int(Nstr,Number),
        concat("Site ",Nstr,Conc1),
        concat(Conc1,": ",Conc2),
        concat(Conc2,Name,Desc).

process(S):-
        pushwords(S,Command,Stat,Sitename,          %--- parses the sentence, determines
                Mon,Year1,Year2),                   %   context, and performs command
        shiftwindow(Old),
        shiftwindow(5),
        determine_context(Command,Stat,Sitename,Mon,Year1,Year2),
        read_context(Com,St,Site,M,Y1,Y2),
        perform(Com,St,Site,M,Y1,Y2),!,
        shiftwindow(Old).
process(_):- shiftwindow(3),nl,
        write("Unable to process this request."),nl.

assert_context(Command,Stat,Sitename,Mon,Year1,Year2):-      %--- asserts current
        asserta(context(command,Command)),                  %   context into DB
        asserta(context(stat,Stat)),
        asserta(context(site,Sitename)),
        asserta(context(month,Mon)),
        asserta(context(year1,Year1)),
        asserta(context(year2,Year2)).

read_context(Command,Stat,Sitename,Mon,Year1,Year2):-        %--- reads context from
        context(command,Command),                           %   DB
        context(stat,Stat),
        context(site,Sitename),
        context(month,Mon),
        context(year1,Year1),
        context(year2,Year2).

determine_context(Command,Stat,Sitename,Mon,Year1,Year2):-
        establish_old_context,
        retractall(context(_,_)),
        establish_new_context(Command,Stat,Sitename,Mon,Year1,Year2).

establish_new_context(Command,none,Sitename,none,none,none):-
        Command <> "plot",!,                                %--- figures out context
        old_context(stat,Stat),                             %   if some keywords
        old_context(month,Mon),                             %   are missing
        old_context(year1,Year1),
        old_context(year2,Year2),
        assert_context(Command,Stat,Sitename,Mon,Year1,Year2).
```

```
establish_new_context(Command,none,Sitename,none,none,none):-
        Command <> "plot",!,
        old_context(stat,Stat),
        old_context(month,Mon),
        old_context(year1,Year1),
        old_context(year2,Year2),
        assert_context(Command,Stat,Sitename,Mon,Year1,Year2).
establish_new_context(Command,none,Sitename,none,Year1,Year2):-
        Command <> "plot",!,
        old_context(stat,Stat),
        old_context(month,Mon),
        assert_context(Command,Stat,Sitename,Mon,Year1,Year2).
establish_new_context(Command,Stat,Sitename,none,none,none):-
        old_context(month,Mon),
        old_context(year1,Year1),
        old_context(year2,Year2),
        assert_context(Command,Stat,Sitename,Mon,Year1,Year2).
establish_new_context(Command,none,Sitename,Mon,none,none):-
        Command <> "list",!,
        old_context(stat,Stat),
        old_context(year1,Year1),
        old_context(year2,Year2),
        assert_context(Command,Stat,Sitename,Mon,Year1,Year2).
establish_new_context(Command,Stat,Sitename,Mon,Year1,Year2):-
        all_or_none(Stat),
        Command = "list",!,
        assert_context(what,all,Sitename,Mon,Year1,Year2).
establish_new_context(Command,Stat,Sitename,Mon,Year1,Year2):-
        Command = "list",!,
        assert_context(what,Stat,Sitename,Mon,Year1,Year2).
establish_new_context(Command,Stat,Sitename,Mon,Year1,Year2):-
        assert_context(Command,Stat,Sitename,Mon,Year1,Year2).

establish_old_context:-
        retractall(old_context(_,_)),
        context(site,OldSitename),
        context(stat,OldStat),
        context(month,OldMon),
        context(year1,OldYear1),
        context(year2,OldYear2),
        asserta(old_context(site,OldSitename)),
        asserta(old_context(stat,OldStat)),
        asserta(old_context(month,OldMon)),
        asserta(old_context(year1,OldYear1)),
        asserta(old_context(year2,OldYear2)).
establish_old_context:- !.

all_or_none(all).
all_or_none(none).
policy_command(yield).
policy_command(loss).
policy_command(policy).

%--- all_or_none defines two
%   possible values, all or none.
%--- these are the keywords for
%   policy commands
```

```
/*----------------------- Parsing the command -------------*/

pushwords(S,Command,Percent,Demand,Mon,Year1,Year2):-      %--- First, if a policy word
        get_command(S,_,Command),                          %    is found then parsing is
        policy_command(Command),!,                         %    a little different
        get_percent(S,_,Percent),
        get_demand(S,_,Demand),
        Year1 = "none",
        Year2 = "none",
        get_month(S,_,Mon).
pushwords(S,Command,Stat,Sitename,Mon,Year1,Year2):-       %--- The normal parse.  Find
        get_command(S,_,Command),                          %    all keywords in a
        get_stat(S,_,Stat),                                %    sentence.
        get_site(S,_,Sitename),
        get_month(S,_,Mon),
        get_year1(S,S1,Year1),
        get_year2(S1,_,Year2).

get_command(S,S2,C):-                                      %--- All these 'get_' clauses
        next_word(S,S2,Com),                               %    find the keywords in the
        word(command,Com,C).                               %    sentence by category.
get_command(S,S2,C):-
        next_word(S,S1,_),!,
        get_command(S1,S2,C).
get_command(S,S,C):- C=what.

get_stat(S,S2,Skey):-
        next_word(S,S2,M),
        word(stat,M,Skey).
get_stat(S,S2,M):-
        next_word(S,S1,_),!,
        get_stat(S1,S2,M).
get_stat(S,S2,M):- S2=S, M=none.

get_percent(S,S2,M):-
        next_word(S,S1,M),
        str_int(M,P),P>29,P<101,
        next_word(S1,S2,Word),
        word(percentword,Word,_).
get_percent(S,S2,Num):-
        next_word(S,S2,Per),
        str_len(Per,Length),
        L2 = Length - 1,
        frontstr(L2,Per,Num,Sign),
        str_int(Num,P),P>29,P<101,
        Sign = "%".
get_percent(S,S2,M):-
        next_word(S,S1,_),!,
        get_percent(S1,S2,M).
get_percent(S,S2,M):- S2=S, M=none.
```

```
get_demand(S,S3,M):-
        next_word(S,S2,Word),
        word(demandword,Word,_),
        get_demand_number(S2,S3,M).
get_demand(S,S2,M):-
        next_word(S,S1,_),!,
        get_demand(S1,S2,M).
get_demand(S,S2,N):- S2=S,context(site,N),
        str_int(N,D),D>80,D<200.
get_demand(S,S2,M):- S2=S, M=none.
get_demand_number(S,S2,Number):-
        next_word(S,S2,Number),
        str_int(Number,N),
        N<200,N>80.
get_demand_number(S,S2,M):-
        next_word(S,S1,_),!,
        get_demand_number(S1,S2,M).

get_site(S,S2,Sitekey):-
        next_word(S,S2,N),
        word(site,N,Sitekey).
get_site(S,S3,Sitekey):-
        next_word(S,S2,Nsite),
        word(sites,Nsite,_),
        get_sitenumber(S2,S3,N),
        symbol_string(Nsite,Place),
        symbol_string(N,Number),
        concat(Place,Number,Sitename),
        word(site,Sitename,Sitekey).
get_site(S,S2,N):-
        next_word(S,S1,_),!,
        get_site(S1,S2,N).
get_site(S,S2,N):- S2=S,context(site,N).
get_site(S,S2,N):- S2=S,N=none.

get_sitenumber(S,S2,Number):-
        next_word(S,S2,Number),
        int(Number).
get_sitenumber(S,S2,N):-
        next_word(S,S2,Number),
        int(Number),
        N=none.

get_month(S,S2,Monthkey):-
        next_word(S,S2,Q),
        word(month,Q,Monthkey).
get_month(S,S2,Q):-
        next_word(S,S1,_),!,
        get_month(S1,S2,Q).
get_month(S,S2,Q):- S2=S,Q=none.
```

```
  get_year1(S,S2,R):-
        next_word(S,S2,R),
        str_int(R,I),
        I > 1900,
        I < 1990.
  get_year1(S,S2,R):-
        next_word(S,S1,_),!,
        get_year1(S1,S2,R).
  get_year1(S,S,R):- R=none.

  get_year2(S,S2,R):-
        next_word(S,S2,R),
        str_int(R,I),
        I > 1900,
        I < 1990.
  get_year2(S,S2,R):-
        next_word(S,S1,_),!,
        get_year2(S1,S2,R).
  get_year2(S,S,R):- R=none.

/*---------------------------- THE PERFORMS ------------------------------*/
%**** These clauses take all the keywords and then perform the command ****

/*---perform(C,M,N,Q,R,T) :-                    %--- Use to check if the correct
        nl,write("the variables are: "),nl,     %    words are being found.
        write(C," ",M," ",N," ",Q," ",R," ",T),
        nl,readchar(_),fail.
----------------*/

  perform(what,Stat,Site,Month,none,none):-
        not(all_or_none(Stat)),
        frontstr(3,Month,C3,_),
        month_integer(C3,Mint),
        findall(F,siteflow(Site,Mint,_,F),L),
        find_stat(Stat,L,Value),
        syn(Stat,Statname),
        syn(C3,Msyn),
        sitename(Site,Sitename),
        units(Stat,Units),
        write("The ",Statname," of the ",Msyn," flows at ",Sitename," was: "),
        writef("%-6.2f ",Value),write(Units),nl.
  perform(what,Stat,Site,none,Year1,Year2):-
        not(all_or_none(Stat)),
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        findall(F,yearf(Site,_,Y1,Y2,F),L),
        find_stat(Stat,L,Value),
        syn(Stat,Statname),
        units(flow,Units),
        write("The ",Statname," of the flows from ",Year1," to ",Year2," is : "),
        writef("%-6.2f ",Value),write(Units),nl.
```

```
perform(what,Stat,Site,_,R,none):-
        not(all_or_none(Stat)),
        str_int(R,I),
        findall(F,siteflow(Site,_,I,F),L),
        find_stat(Stat,L,Value),
        syn(Stat,Statname),
        sitename(Site,Sitename),
        units(Stat,Units),
        write("The ",Statname," for ",I," at ",Sitename," was: "),
        writef("%-6.2f ",Value),write(Units),nl.
perform(what,Stat,Site,Month,none,none):-
        all_or_none(Stat),
        frontstr(3,Month,C3,_),
        month_integer(C3,Mint),
        findall(F,siteflow(Site,Mint,_,F),L),
        sitename(Site,Sitename),
        units(flow,Units),
        syn(C3,Msyn),
        write("The ",Msyn," flows for ",Sitename," are ",Units),nl,
        writeflist(L),nl.
perform(what,Stat,Site,Month,Year1,Year2):-
        all_or_none(Stat),
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        frontstr(3,Month,C3,_),
        month_integer(C3,Mint),
        syn(C3,Msyn),
        findall(F,yearf(Site,Mint,Y1,Y2,F),L),
        sitename(Site,Sitename),
        units(flow,Units),
        write("The ",Msyn," flows for ",Sitename," are ",Units),nl,
        nl,writeflist(L).
perform(what,Stat,Site,Month,Year1,Year2):-
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        frontstr(3,Month,C3,_),
        month_integer(C3,Mint),
        syn(C3,Msyn),
        findall(F,yearf(Site,Mint,Y1,Y2,F),L),
        find_stat(Stat,L,Value),
        sitename(Site,Sitename),
        units(Stat,Units),
        write("The ",Stat," for ",Msyn," from ",Y1," to ",Y2," at ",Sitename," is "),
        writef("%-6.2f ",Value),write(Units),nl.
perform(what,Stat,Site,none,Year1,Year2):-
        all_or_none(Stat),
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        findall(F,yearf(Site,_,Y1,Y2,F),L),
        sitename(Site,Sitename),units(flow,Units),
        write("The flows at ",Sitename," from ",Year1," to ",Year2," are ",Units),nl,
        nl,writeflist(L).
```

```
perform(what,Stat,Site,none,Year,none):-
        all_or_none(Stat),
        str_int(Year,Y),
        findall(F,siteflow(Site,_,Y,F),L),
        sitename(Site,Sitename),
        units(flow,Units),
        write("The flows at ",Sitename," for ",Year," are ",Units),nl,
        writeflist(L),nl.
perform(what,_,Site,Month,Year,none):-
        str_int(Year,Y),
        frontstr(3,Month,C3,_),
        month_integer(C3,Mint),
        syn(C3,Msyn),
        findall(F,siteflow(Site,Mint,Y,F),L),
        head(Fl,L),
        sitename(Site,Sitename),
        units(flow,Units),
        write("The flow at ",Sitename," for ",Msyn," of ",Year," was ",Fl,Units),nl.

perform(plot,cdf,Site,Xmonth1,none,none):-          %--- the plotting performs
        Fore = 63,
        Back = 1,
        frontstr(3,Xmonth1,M1,_),
        syn(M1,Msyn),
        sitename(Site,Sitename),
        concat("CDF for ",Msyn,T1),
        concat(T1," at ",T2),
        concat(T2,Sitename,Title),
        month_integer(M1,Mint),
        findall(F,siteflow(Site,Mint,_,F),L),
        Xaxis = "              Year",Yaxis = "STREAMFLOW cfs",
        quicksort(L,SortedL),
        plot_list1(SortedL,Title,Xaxis,Yaxis,monthly,1,Fore,Back).
perform(plot,cdf,Site,none,Year1,none):-
        Fore = 63,
        Back = 1,
        str_int(Year1,Y1),
        sitename(Site,Sitename),
    concat(" CDF for ",Year1,T1),
        concat(T1," at ",T2),
    concat(T2,Sitename,Title),
        findall(F,siteflow(Site,_,Y1,F),L),
        Xaxis = "         Months",Yaxis = "STREAMFLOW cfs",
        quicksort(L,SortedL),
        plot_list1(SortedL,Title,Xaxis,Yaxis,monthly,1,Fore,Back).
```

```
perform(plot,cdf,Site,Xmonth1,Year1,Year2):-
        Fore = 63,
        Back = 1,
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        frontstr(3,Xmonth1,M1,_),
        syn(M1,Msyn),
        concat("CDF for ",Msyn,Title),
        month_integer(M1,Mint),
        Xaxis = "          Year",Yaxis = "STREAMFLOW cfs",
        findall(F,yearf(Site,Mint,Y1,Y2,F),L),
        quicksort(L,SortedL),
        plot_list1(SortedL,Title,Xaxis,Yaxis,monthly,1,Fore,Back).
perform(plot,Stat,Site,Xmonth1,none,none):-
        Fore = 63,
        Back = 1,
        all_or_none(Stat),
        frontstr(3,Xmonth1,M1,_),
        syn(M1,Msyn),
        sitename(Site,Sitename),
        concat(Msyn," flows for all years on record at ",T1),
        concat(T1,Sitename,Title),
        month_integer(M1,Mint),
        findall(F,siteflow(Site,Mint,_,F),L),
        Xaxis = "          Year",Yaxis = "Streamflow cfs",
        plot_list1(L,Title,Xaxis,Yaxis,monthly,1929,Fore,Back).
perform(plot,Stat,Site,none,Year1,none):-
        Fore = 63,
        Back = 1,
        all_or_none(Stat),
        str_int(Year1,Y1),
        findall(F,siteflow(Site,_,Y1,F),L),
        Smtitle = " Monthly flows for the year ",
        concat(Smtitle,Year1,Title),
        Xaxis = Year1,Yaxis = "Streamflow cfs",
        plot_list1(L,Title,Xaxis,Yaxis,oneyear,1,Fore,Back).
perform(plot,Stat,Site,Xmonth1,Year1,Year2):-
        Fore = 63,
        Back = 1,
        all_or_none(Stat),
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        frontstr(3,Xmonth1,M1,_),
        month_integer(M1,Mint),
        findall(F,yearf(Site,Mint,Y1,Y2,F),L),
        syn(M1,Title),
        Xaxis = "          Year",Yaxis = "Streamflow cfs",
        plot_list1(L,Title,Xaxis,Yaxis,monthly,Y1,Fore,Back).
```

```
perform(plot,Stat,Site,none,Year1,Year2):-
        Fore = 63,
        Back = 1,
        all_or_none(Stat),
        str_int(Year1,Y1),
        str_int(Year2,Y2),
        seq_yearf(Site,Y1,Y2,L),
        sitename(Site,Sitename),
        T1 = " All Monthly Streamflows at ",
        concat(T1,Sitename,Title),
        Xaxis = "          Year",Yaxis = "Streamflow cfs",
        plot_list1(L,Title,Xaxis,Yaxis,allmonths,Y1,Fore,Back).

perform(Policy,_,none,_,_,_):-                    %--- the policy performs
        policy_command(Policy),!,
        nl,write("Either the demand is out of range or demand was not specified.").
perform(Policy,none,_,_,_,_):-
        policy_command(Policy),!,
        nl,write("Please specify an initial storage level between 30 to 100%").

perform(yield,Percent,Demand,Mon,_,_):-
        Fore = 63,
        Back = 1,
        readdevice(Current_device),
        run_policy(Percent,Demand,Mon),
        read_policy(Ylist,_,_),
        readdevice(Current_device),
        Title = "  Yields for a 16 week period",
        Xaxis = "       Year Number",Yaxis = "Yield cfs",
        plot_list1(Ylist,Title,Xaxis,Yaxis,monthly,1,Fore,Back).
perform(loss,Percent,Demand,Mon,_,_):-
        Fore = 63,
        Back = 1,
        readdevice(Current_device),
        run_policy(Percent,Demand,Mon),
        read_policy(_,Llist,_),
        readdevice(Current_device),
        Title = "  Economic losses for a 16 week period",
        Xaxis = "       Year Number",Yaxis = "Loss",
        plot_list1(Llist,Title,Xaxis,Yaxis,monthly,1,Fore,Back).
perform(policy,Percent,Demand,Mon,_,_):-
        readdevice(Current_device),
        run_policy(Percent,Demand,Mon),
        read_policy(_,_,Oplist),
        readdevice(Current_device),
        write("The years that had economic losses associated with these conditions are:"),
        nl,writeheads(Oplist),nl,
        write("Enter year number to see optimal operation policies."),nl,
        shiftwindow(3),readln(S),
        str_int(S,Sint),
        nl,members(Sint,Oplist,Polist),shiftwindow(5),
        barchart(Polist,S,Mon).
```

```
perform(Policy,_,_,_,_,_):-
        policy_command(Policy),!,
        nl,write("No record exists in the database for these conditions.").

                                                        %--- miscellaneous performs

perform(help,_,_,_,_,_):-
        write("All sentences must be followed with either a period or a '?'."),nl,
        write("A sample sentence may be composed of the following types of keywords:"),nl,
        write("<command><statistic><site name><month><year 1><year 2>"),nl,nl,
        write("Do you want to see a list of the words available? (y/n) "),
        readln(Ans),!,
        upper_lower(Ans,Answer),
        list_commands(Answer).
perform(select,B,_,D,E,F):-
        select_site(Site),!,
        old_context(command,A),
        perform(A,B,Site,D,E,F).
perform(memory,_,_,_,_,_):-show_memory.

perform(A,B,none,D,E,F):- !,nl,                 %--- catch the rest of the
        write("Either a site has not been choosen or "),    %   combinations of keywords
        nl,
        write("the site number entered is not a valid site."),
        nl,select_site(Site),
        perform(A,B,Site,D,E,F).

perform(_,_,_,_,_,_):- nl,!,
        write("No form for this sentence. Please reword."),nl,
        write("Type 'help.'for help"),nl.


run_policy(Percent,Demand,Mon):-                %--- Runs the POLICY.EXE code
        readdevice(Current_device),             %   from DOS to get the policy
        openwrite(polfile,"code.in"),           %   data given a specific situation
        writedevice(polfile),
        getpolicy(Percent,Demand,Mon,PolicyCode),
        write(PolicyCode),
        closefile(polfile),
        openwrite(garbage,"garbage.dat"),
        writedevice(garbage),
        system("Policy",0,DosErrorLevel),
        write(DosErrorLevel),
        closefile(garbage),
        readdevice(Current_device),
        writedevice(screen).
```

```
read_policy(Yield,Loss,Oplist):-                    %--- Reads the data file created by
        readdevice(Current_device),                 %    POLICY.EXE
        openread(codeout,"code.out"),
        readdevice(codeout),
        scan_lines(5,Yield),
        scan_lines(5,Loss),
        read_years(Backlist,[],codeout),
        reverse_lists(Backlist,Oplist),
        closefile(codeout),
        readdevice(Current_device).

read_years(Oplist,Sublist,Fname):-                  %--- Reads the years which had
        not(eof(Fname)),!,                          %    economic losses given the
        readln(Line),                               %    current situation
        scanner_int(Line,Ylist),
        addlists(Ylist,Sublist,Newoplist),
        read_years(Oplist,Newoplist,Fname).
read_years(Oplist,Oplist,_).

scan_lines(Nlines,Olist):-                          %--- scans string and makes a list
        scan_lines_aux(Nlines,Olist,1,[]).          %    out of the tokens
scan_lines_aux(Nlines,Olist,Icount,Nlist):-
        Icount<=Nlines,!,
        readln(Line),
        scanner(Line,Sublist),
        append(Nlist,Sublist,Newlist),
        NewIcount = Icount + 1,
        scan_lines_aux(Nlines,Olist,NewIcount,Newlist).
scan_lines_aux(Nlines,Olist,Icount,Olist):- Icount > Nlines.

scanner("",[]).                                     %--- scans string for tokens, which
scanner(Str,[Tok1|Rest]):-                          %    in this case are real numbers
        fronttoken(Str,Sym,Str1),!,
        str_real(Sym,Tok1),
        scanner(Str1,Rest).

scanner_int("",[]).                                 %--- does the same for integers
scanner_int(Str,[Tok1|Rest]):-
        fronttoken(Str,Sym,Str1),!,
        str_int(Sym,Tok1),
        scanner_int(Str1,Rest).

getpolicy(Store,Demand,Mon,PolCode):-               %--- makes the policy code ascii
        syn(Mon,Month),                             %    file required by POLICY.EXE
        frontstr(3,Month,M1,_),
        concat("C",M1,S),
        concat(S,"1",S1),
        concat(S1,".",S2),
        concat(S2,Store,S3),
        concat(S3,".",S4),
        concat(S4,Demand,PolCode).
```

```
find_delim(S,Count,C):-                          %--- peels delimiters off the
        frontchar(S,CH,S2),                      %    end of a token (word).
        CH<>' ',CH<>'.',CH<>'?',CH<>',',
        C2=C+1,
        find_delim(S2,Count,C2).
find_delim(_,Count,Count).

get_noise(S,S2,X):-                              %--- gets rid of noise or words not
        next_word(S,S2,X),                       %    defined in the lexicon
        not(word(_,X,_)),
        not(int(X)).

terminator(S):-                                  %--- finds the terminator of input,
        frontchar(S,CH,_),                       %    not required, in this code
        CH='.'.
terminator(S):-
        frontchar(S,CH,_),
        CH='?'.

next_word(S,S2,W):-
        find_delim(S,Count,0),!,
        Count>0,
        frontstr(Count,S,W,S3),
        strip_space(S3,S2).

next_valid_word(S,S2,W):-                        %--- find the next keyword or number
        find_delim(S,Count,0),!,                 %    in the input string
        Count>0,
        frontstr(Count,S,W,S3),
        word_or_number(W),
        strip_space(S3,S2).
next_valid_word(S,S2,W):-
        get_noise(S,S1,_),
        next_valid_word(S1,S2,W).

word_or_number(W):-!,word(_,W,_).                %--- Is it a word defined in the lexicon?
word_or_number(W):-!,int(W).                     %--- Is it a number?

strip_space(S,S2):-
        frontchar(S,Ch,S3),
        Ch=' ',!,
        strip_space(S3,S2).
strip_space(S,S2):-
        frontchar(S,Ch,S3),
        Ch=',',!,
        strip_space(S3,S2).
strip_space(S,S2):-
        frontchar(S,Ch,S3),
        Ch=':',!,
        strip_space(S3,S2).
strip_space(S,S).
```

```
find_stat(Stat,List,Value):-                    %--- takes a list and stat
        Stat=average,!,                         %    and then finds the value
        length_of(List,N),meanlist(List,Value,N).   %    of that stat.
find_stat(Stat,List,Value):-
        Stat="standard",!,
        length_of(List,N),standardList(List,Value,N).
find_stat(Stat,List,Value):-
        Stat=skew,!,
        length_of(List,N),skew(List,Value,N).
find_stat(Stat,List,Value):-
        Stat=variance,!,
        length_of(List,N),varian(List,Value,N).
find_stat(Stat,List,Value):-
        Stat=lowest,!,
        minimum(Value,List).
find_stat(Stat,List,Value):-
        Stat=highest,!,
        maximum(Value,List).
find_stat(Stat,List,Value):-
        Stat=sum,!,
        sumlist(List,Value).

units(flow,"(cfs)").                            %--- defining units for printing
units(average,"(cfs)").
units(_,"").

/*------------------------------------ the help screen ------------------------*/

list_commands(Answer):-nl,frontchar(Answer,A,_),A='y',!,
        write("  Commands   Statistics     Sites          Months       Years"),nl,
        write("  ------------------------------------------------------------------"),nl,
        write("  what       average       Cedar 1        January      1929  "),nl,
        write("  list       standard (dev.) Cedar 2      February     to 1976 "),nl,
        write("  plot       skew          Cedar 3        March    "),nl,
        write("  help       variance      Lake Washington  April"),nl,
        write("  show       all           Tolt 1         May"),nl,
        write("  select(site) lowest      Tolt 2         June"),nl,
        write("             highest       Main Stem Tolt   July"),nl,
        write("             cdf               Inflow 1       August"),nl,
        write("             sum           Inflow 2       etc."),nl,
        write("                               Inflow 3           "),nl.
list_commands(_).

purge:-retractall(_),fail.                      %--- purge all databases
purge:-retractall(_,record),fail.
purge:-retractall(_,current_context),fail.
purge.
```

```
/*-------------------------------- NLP_INC.PRO ------------------------------*/
%**** this file includes all the toolbox files needed.

%project "nlp"

include "c:\\prolog\\toolbox\\gdoms.pro"
include "c:\\prolog\\toolbox\\gglobs.pro"
include "globdef.pro"

database                                        %--- this DB is necessary for plotting
        Scale(ScaleNo,x,x,y,y)
        activeScale(ScaleNo)
        axes(Integer,Integer,Integer,Xmarker,Ymarker,Col,Row,Col,Row)
        insmode
        lineinpstate(STRING,COL)
        lineinpflag

include "c:\\prolog\\toolbox\\tpreds.pro"
include "c:\\prolog\\toolbox\\gpreds.pro"
include "c:\\prolog\\toolbox\\ggraph2.pro"
include "c:\\prolog\\toolbox\\menu.pro"
include "c:\\prolog\\toolbox\\status.pro"

domains
        sentence = string
        list     = symbol*
        flows    = real
        flowlist = flows*
        listlist = flowlist*
        ilists   = integerlist*
        siteName = symbol
        year     = integer
        month    = symbol
        fname    = string

include "c:\\prolog\\toolbox\\lineinp.pro"
```

```
/*------------------------------ NLP_DB.PRO ------------------------------*/
%***** contains clauses associated with the streamflow and lexicon databases

%project "nlp"
database - record
   word(symbol,symbol,symbol)
   syn(symbol,symbol)
   month_integer(symbol,integer)
   sitenumber(integer,symbol)
   siten(integer,symbol)
   sitename(symbol,symbol)
   site1(integer,year,flows)
   site5(integer,year,flows)
   site7(integer,year,flows)
   site8(integer,year,flows)
   site11(integer,year,flows)
   site15(integer,year,flows)
   site16(integer,year,flows)

predicates
/*--------------- Database Predicates ---------------*/
    consultFiles
    consultWords
    my_consult(string)
    repfile(file)

clauses
/*--------------- data bases ----------------------*/
        consultFiles:-
                my_consult("site1.dat"),
                my_consult("site5.dat"),
                my_consult("site7.dat"),
                my_consult("site8.dat"),
                my_consult("site11.dat"),
                my_consult("site15.dat"),
                my_consult("site16.dat").
        consultWords:- my_consult("words.dat"),
                my_consult("words2.dat").

        my_consult(FileName):-                  %--- open files and assert
                openread(datafile,FileName),    %    facts
                readdevice(datafile),
                repfile(datafile),
                readterm(record,Term),
                assertz(Term),
                fail.
        my_consult(_):- eof(datafile),!,closefile(datafile).

        repfile(_).
        repfile(File) :- not(eof(File)), repfile(File).
```

```
/*---------------------------- NLP_FLOW.PRO -----------------------------*/
%**** the streamflow and list handling predicates.

%project "nlp"
predicates
        writeheads(ilists)
        siteflow(symbol,integer,year,flows)
        writeflist(flowlist)
        write5(flowlist,integer)
        seq_yearf(symbol,year,year,flowlist)
        yearf(symbol,integer,year,year,flows)

clauses

/*--------------------- Seting Up Data ----------------------*/
        siteflow(site1,Mon,Year,Flow):-    !,site1(Mon,Year,Flow).
        siteflow(site5,Mon,Year,Flow):-    !,site5(Mon,Year,Flow).
        siteflow(site7,Mon,Year,Flow):-    !,site7(Mon,Year,Flow).
        siteflow(site8,Mon,Year,Flow):- !,site8(Mon,Year,Flow).
        siteflow(site11,Mon,Year,Flow):- !,site11(Mon,Year,Flow).
        siteflow(site15,Mon,Year,Flow):- !,site15(Mon,Year,Flow).
        siteflow(site16,Mon,Year,Flow):- !,site16(Mon,Year,Flow).
/*****************************************************************/

        writeflist(List):-write5(List,0),nl.          %--- write flow list.

        write5(List,12):- !, nl, write5(List,0).      %--- write list 12 across.
        write5([Head|Tail],N):- !,
                writef("%-5.1f ",Head),
                N1=N+1,write5(Tail,N1).
        write5([],_).

        writeheads([[Head|_]|Tlist]):- !,
                write(" ",Head),
                writeheads(Tlist).
        writeheads([]).

        yearf(Site,Month,Year1,Year2,F):-             %--- make list between two
                siteflow(Site,Month,Y,F),             %    years, for a particular month.
                Y<=Year2,Y>=Year1.

        seq_yearf(Site,Year2,Year2,Rlist):-!,         %--- make list for all months
                findall(F,siteflow(Site,_,Year2,F),Rlist).
        seq_yearf(Site,Year1,Year2,L):-Year1<>Year2,
                Y2=Year2-1,
                seq_yearf(Site,Year1,Y2,List),!,
                findall(F,siteflow(Site,_,Y2,F),AList),
                append(AList,List,L).
```

```
/*---------------------------------- NLP_STAT.PRO ----------------------------*/
%**** statistical and list clauses

%project "nlp"
/*--------------------- Statistics --------------------*/
predicates
    list_existing_windows(integer)
    length_of(flowlist,integer)
    find_stat(symbol,flowlist,flows)
    meanlist(flowlist,flows,integer)
    sumlist(flowlist,flows)
    sumlist1(flowlist,flows,flows)
    standardList(flowlist,flows,integer)
    var1(flows,flows,flows,flowlist)
    varian(flowlist,real,integer)
    sumMinusMean(flowlist,flows,flows)
    skew(flowlist,flows,integer)
    skew1(flows,flows,flows,flows,flowlist)
    cube(flows,flows)
    quicksort(flowlist,flowlist)
    split(flows,flowlist,flowlist,flowlist)
    append(flowlist,flowlist,flowlist)
 /*-------------------------------------------------------------------------
          the s-notation distinguishes predicates that operate on lists of lists
          ie.  members(_,_).
 -------------------------------------------------------------------------*/
          members(integer,ilists,integerlist)
          n_element(integerlist,integer,integer)
          reverse_lists(ilists,ilists)
          reverses(ilists,ilists,ilists)
          int_real(integer,real)
          ints_reals(integerlist,flowlist)
          int(symbol)
          add(flows,flowlist,flowlist)
          addlists(integerlist,ilists,ilists)
          head(flows,flowlist)
          maximum(integer,flowlist)
          maxi(integer,integer,flowlist)
          minimum(integer,flowlist)
          mini(integer,integer,flowlist)

clauses
/*------------      Many of these statistical and list clauses are from
                   Townsend, pp 70-90 ----------------------------------------*/

  minimum(Minimum,[Head|Tail]):-                %--- find minimum element
          mini(Minimum,Head,Tail).
  mini(Start,Start,[]).
  mini(End,Start,[Head|Tail]):-
          Head <= Start,
          mini(End,Head,Tail).
```

```
mini(End,Start,[Head|Tail]):-
        Head >= Start,
        mini(End,Start,Tail).

maximum(Maximum,[Head|Tail]):-          %--- find maximum element
        maxi(Maximum,Head,Tail).
maxi(Start,Start,[]).
maxi(End,Start,[Head|Tail]):-
        Head >= Start,
        maxi(End,Head,Tail).
maxi(End,Start,[Head|Tail]):-
        Head <= Start,
        maxi(End,Start,Tail).

reverse_lists(Inlist,Outlist):-         %--- reverses a list of lists
        reverses(Inlist,[],Outlist).
reverses([],Inlist,Inlist):-!.
reverses([Head|Tail],List1,List2):-
        reverses(Tail,[Head|List1],List2).

members(Element,[[Element|Tail]|_],Tail):-!.    %--- Determines if element is
members(Element,[_|Tail],Result):-              %   the head of one of the lists
        members(Element,Tail,Result).           %   and then returns the assoc tail.

n_element([Head|_],1,Element):-         %--- At what position in the list
        Element=Head,!.                 %   is this element?
n_element([_|Tail],No,Element):-
        NN=No-1,!,
        n_element(Tail,NN,Element).

int(X):-str_int(X,I),I=I.               %--- check to see if the string object is
                                        %   an integer
int_real(X,Y):-Y=X.                     %--- convert an integer to a real or vice versa

ints_reals([],[]):-!.                   %--- converts a list of integers to a list
ints_reals([X|T],[Y|U]):-               %   of reals
        int_real(X,Y),
        ints_reals(T,U).

append([],L,L).                         %--- appends a list to the end of another
append([X|L1],L2,[X|L3]) :-             %   from Turbo Prolog 1.0 manual pg. 49
        append(L1,L2,L3).

head(X,[X|_]).
add(X,[],[X]).                          %--- add an object to the front
add(X,L,[X|L]).                         %   of a list

addlists(X,[],[X]).                     %--- add a list to the front
addlists(X,L,[X|L]).                    %   of another list
```

```
quicksort([],[]).                          %--- used to rank, (for cdf)
quicksort([X|Tail],Sorted) :-
        split(X,Tail,Small,Big),
        quicksort(Small,SortedSmall),
        quicksort(Big,SortedBig),
        append(SortedSmall,[X|SortedBig],Sorted).

split(X,[],[],[]):- X=X.                   %--- used to sort and rank
split(X,[Y|Tail],[Y|Small],Big) :-
        X>Y,!,
        split(X,Tail,Small,Big).
split(X,[Y|Tail],Small,[Y|Big]) :-
        split(X,Tail,Small,Big).

meanlist(X,A,N):-                          %--- finds the mean of a list
        sumlist(X,Sum),A=Sum/N.

sumlist([Head|Tail],Sum):-                 %--- sums a list
        sumlist1([Head|Tail],0,Sum),!.
sumlist1([Head|Tail],X,Sum):-
        Temp = X + Head,
        sumlist1(Tail,Temp,Sum).
sumlist1([],Sum,Sum).

sumMinusMean([I|Is],Mean,Sum):- sumMinusMean(Is,Mean,IsSum),
        Sum=abs(I-Mean)*abs(I-Mean)+IsSum.
sumMinusMean([],_,0).

standardList(List,Standard,N):-            %--- finds standard deviation
        varian(List,Variance,N),
        Standard = sqrt(Variance),!.
varian(List,Variance,N):-                  %--- finds variance
        meanlist(List,Mean,N),
        length_of(List,Count),
        var1(Sum,0,Mean,List),
        Variance = Sum / Count,!.
var1(Sum,Sum,_,[]) :- !.
var1(Sum,Running,Mean,[Head|Tail]):-
        Newrunning = Running + (Head - Mean) * (Head - Mean),
        var1(Sum,Newrunning,Mean,Tail),!.

skew(List,Skew,N):-                        %--- finds the skew
        meanlist(List,Mean,N),
        standardList(List,Standard,N),
        skew1(Sum,0,Mean,Standard,List),
        Skew=Sum / N,!.
skew1(Sum,Sum,_,_,[]):-!.
skew1(Sum,Running,Mean,Standard,[Head|Tail]):-
        Expr = (Head - Mean) / Standard,
        cube(Expr,Exp3),
        Newrunning = Running + Exp3,
        skew1(Sum,Newrunning,Mean,Standard,Tail),!.
```

```
  cube(Expr,Cube):- Cube = Expr * Expr * Expr.

  length_of([],0).                          %--- from user manual pg. 187
  length_of([_|T],L):-
        length_of(T,TailLength),
        L = TailLength + 1.

  list_existing_windows(0).                 %--- use for debuging, lists window
  list_existing_windows(N):-                %    numbers for all open windows
        not(existwindow(N)),!,
        N1 = N - 1,
        list_existing_windows(N1).
  list_existing_windows(N):-
        existwindow(N),
        shiftwindow(N),
        write(N," "),readchar(_),
        N1 = N - 1,
        list_existing_windows(N1)./*---------------------------------- NLP_PLOT.PRO ----------
--------*/
%**** plotting clauses

%project "NLP"
%include "c:\\prolog\\toolbox\\gbar.pro"

/*---------------------- Plotting --------------------------*/
predicates
        function(flowlist,real,integer,integer,real,integer,integer)
        plot_list1(flowlist,string,string,string,string,integer,integer,integer)
        determine_Xscale(string,integer,integer,integer,integer,integer,real)
        determine_yscale(integer,integer,real,real,integer)
        barchart(integerlist,string,string)

clauses
/*------------------------------- plots a list -------------------------*/
   plot_list1(List,Title,Xlabel,Ylabel,TimeFrame,YearInit,Fore,Back):-
        maximum(Ymax,List),
        minimum(Ymin,List),
        length_of(List,Length),
        determine_Xscale(TimeFrame,YearInit,Length,Xmin,Xmax,Xunit,SFactor),
        determine_Yscale(Ymin,Ymax,Yminr,Ymaxr,Yunit),
        graphics(5,Fore,Back),
        makewindow(10,Back,Fore,Title,0,0,22,79),
        makestatus(7,"Enter <CR> to return to menu."),
        attribute(Fore),
        Xbegin = Xmin - 1,
        defineScale(1,Xbegin,Xmax,Yminr,Ymaxr),
        makeAxes(1,_,GWindow,
                marker(Xunit,d,4),
                marker(Yunit,d,4),3,4,3,2,Fore),
        Gwindow=Gwindow,
        axisLabels(1,Xlabel,Ylabel),
        function(List,0.0,0,Length,SFactor,YearInit,Fore),
```

```
        readln(_),
        removewindow(10,0),
        text,
        removewindow(20,0).

plot_list1(_,_,_,_,_,_,_,_) :- !,text.

        determine_Xscale(TimeFrame,YearInit,Length,Xmin,Xmax,Xunit,SFactor):-
                TimeFrame="allmonths",!,
                SFactor=1.0/12.0,
                Xunit=1,
                Xmin=YearInit,
                XLength=Length div 12,
                Xmax=XLength+YearInit.
```

```
determine_Xscale(TimeFrame,YearInit,Length,Xmin,Xmax,Xunit,SFactor):-
        Length < 20,
        TimeFrame="monthly",!,
        SFactor=1.0,
        Xunit=2,
        Xmin=YearInit,
        Xmax=Length+YearInit.
determine_Xscale(TimeFrame,YearInit,Length,Xmin,Xmax,Xunit,SFactor):-
        TimeFrame="monthly",!,
        SFactor=1.0,
        Xunit=5,
        Xmin=YearInit,
        Xmax=Length+YearInit.
determine_Xscale(TimeFrame,_,_,Xmin,Xmax,Xunit,SFactor):-
        TimeFrame="oneyear",
        SFactor=1.0,
        Xunit=2,
        Xmin=1,
        Xmax=12.

determine_yscale(Ymin,Ymax,Yminr,Ymaxr,Yunit):-
        Yrange=Ymax-Ymin,
        Y=Yrange/100,
        Y>9,!,
        Yminr=(Ymin div 100)*100.0,
        Ymaxr=(Ymax div 100)*100.0 + 100.0,
        Yunit=100.
determine_yscale(Ymin,Ymax,Yminr,Ymaxr,Yunit):-
        Yrange=Ymax-Ymin,
        Y=Yrange/100,
        Y<=9,!,
        Yminr=(Ymin div 50)*50.0,
        Ymaxr=(Ymax div 50)*50.0 + 50.0,
        Yunit=50.

function(_,_,N,Length,_,_,_):-                  %--- function plots one segment at
        L2 = Length - 2,                        %    a time
        N > L2,!.
function([Head|Tail],Y1,N,Length,SFactor,YearInit,Color) :-
        Y1=0.0,!,
        N1 = N + 1,
        YY1=Head/1.0001,
        X1=N/1.0001+YearInit,
        X2=N1/1.0001*SFactor+YearInit,
        head(Head2,Tail),
        Y2=Head2/1.0001,
        scaleLine(X1,YY1,X2,Y2,Color),
        function(Tail,Y2,N1,Length,SFactor,YearInit,Color).
```

```
        function([Head|Tail],Y1,N,Length,SFactor,YearInit,Color):-
                N1 = N + 1,!,
                X1=N/1.0001*SFactor+YearInit,
                X2=N1/1.0001*SFactor+YearInit,
                Y1=Head/1.0001,
                head(Head2,Tail),
                Y2=Head2/1.0001,
                scaleLine(X1,Y1,X2,Y2,Color),
                function(Tail,Y2,N1,Length,SFactor,YearInit,Color).
/*************************************************************************
        BAR CHART of OPTIMAL POLICIES (modified, not really a bar now)
*************************************************************************/

barchart(Oplist,Year,Month):-
 Fore = 63,
 Back = 1,
 concat("OPTIMAL RESERVIOR OPERATION POLICY FOR SUMMER OF ",Year,Title),
 graphics(5,Fore,Back),
 makewindow(10,Back,Fore,Title,0,0,22,79),
 makestatus(7,"Enter <CR> to return to menu."),
 attribute(Fore),
 concat("Weeks After ",Month,Substr),
 concat(Substr," 1st",Subtitle),
 Xlabel = Subtitle,
 Ylabel = "STAGE",
 ints_reals(Oplist,List),
 Ymaxr = 5,
 Yminr = 0,
 length_of(List,Length),
 defineScale(1,0,16,Yminr,Ymaxr),
 makeAxes(1,_,GWindow,
                marker(1,d,4),
                marker(1,d,4),3,4,3,2,Fore),
 Gwindow=Gwindow,
 axisLabels(1,Xlabel,Ylabel),
 function(List,0.0,0,Length,1,1,Fore),
 readln(_),
 removewindow(10,0),
 text,
 removewindow(20,0).
```

```
/*---------------------------------------------- WORDS.DAT and WORDS2.DAT----------------------*/
%*****    THE LEXICON
%
word("site","site1","site1")
word("site","site5","site5")
word("site","site7","site7")
word("site","site8","site8")
word("site","site11","site11")
word("site","site15","site15")
word("site","site16","site16")
word("site","inflow1","site1")
word("site","inflow2","site5")
word("site","inflow3","site7")
word("site","tolt1","site11")
word("site","tolt2","site15")
word("site","tolt3","site16")
word("site","lake","site8")
word("site","north","site15")
word("site","south","site16")
word("site","main","site11")
word("site","cedar1","site1")
word("site","cedar2","site5")
word("site","cedar3","site7")
word("site","cedar4","site8")
word("command","forget","forget")
word("command","clear","forget")
word("command","not","forget")
word("command","memory","memory")
word("command","resize","resize")
word("command","plot","plot")
word("command","help","help")
word("command","list","list")
word("command","what","what")
word("command","show","what")
word("command","select","select")
word("command","yield","yield")
word("command","loss","loss")
word("command","losses","loss")
word("command","economic","loss")
word("command","yields","yield")
word("command","policy","policy")
word("command","policies","policy")
word("command","operating","policy")
word("command","operation","policy")
word("stat","cdf","cdf")
word("stat","average","average")
word("stat","mean","average")
word("stat","standard","standard")
word("stat","variance","variance")
word("stat","skew","skew")
word("stat","all","all")
word("stat","sum","sum")
```

```
word("demandword","demand","demand")
word("percentword","percent","%")
word("percentword","%","%")
word("month","summer","summer")
word("stat","low","low")
word("stat","lowest","lowest")
word("stat","minimum","lowest")
word("stat","high","highest")
word("stat","highest","highest")
word("stat","maximum","highest")
word("stats",">","greater than")
word("stats","<","less than")
word("stats",">=","greater than or equal to")
word("stats","<=","less than or equal to")
word("stat","cdf","cdf")
word("month","january","january")
word("month","february","february")
word("month","march","march")
word("month","april","april")
word("month","may","may")
word("month","june","june")
word("month","july","july")
word("month","august","august")
word("month","september","september")
word("month","october","october")
word("month","november","november")
word("month","december","december")
word("month","summer","summer")
word("month","jan","january")
word("month","feb","february")
word("month","mar","march")
word("month","apr","april")
word("month","may","may")
word("month","jun","june")
word("month","jul","july")
word("month","aug","august")
word("month","sep","september")
word("month","oct","october")
word("month","nov","november")
word("month","dec","december")
word("conj","and","and")
word("sites","site","unknown")
word("sites","cedar","unknown")
word("sites","tolt","unknown")
word("sites","inflow","unknown")
word("stats","lowest","unknown")
word("stats","highest","unknown")
syn("january","January")
syn("february","February")
syn("march","March")
syn("april","April")
syn("may","May")
```

```
syn("june","June")
syn("july","July")
syn("august","August")
syn("september","September")
syn("october","October")
syn("november","November")
syn("december","December")
syn("jan","January")
syn("feb","February")
syn("mar","March")
syn("apr","April")
syn("may","May")
syn("jun","June")
syn("jul","July")
syn("aug","August")
syn("sep","September")
syn("oct","October")
syn("nov","November")
syn("dec","December")
syn("memory","memory")
syn("all","all")
syn("cdf","cummulative distribution function")
syn("standard","standard deviation")
syn("skew","skew")
syn("mean","average")
syn("average","average")
syn("variance","variance")
syn("sum","sum")
syn("summer","sum of June, July and August")
syn("lowest","lowest")
syn("highest","highest")
syn("low","lowest")
syn("high","highest")
syn("inflow","site")
syn("cedar","site")
syn("tolt","site")
syn("what","what")
syn("show","what")
syn("list","what")
syn("plot","plot")
syn("memory","memory")
syn("inflow1","site1")
syn("inflow2","site5")
syn("inflow3","site7")
syn("lake","site8")
syn("main","site11")
syn("north","site15")
syn("south","site16")
syn("cedar1","site1")
syn("cedar2","site5")
syn("cedar3","site7")
syn("inflow1","site1")
```

```
syn("inflow2","site5")
syn("inflow3","site7")
syn("tolt1","site11")
syn("tolt2","site15")
syn("tolt3","site16")
syn("site1","site1")
syn("site5","site5")
syn("site7","site7")
syn("site8","site8")
syn("site11","site11")
syn("site15","site15")
syn("site16","site16")month_integer("jan",1).
month_integer("feb",2).
month_integer("mar",3).
month_integer("apr",4).
month_integer("may",5).
month_integer("jun",6).
month_integer("jul",7).
month_integer("aug",8).
month_integer("sep",9).
month_integer("oct",10).
month_integer("nov",11).
month_integer("dec",12).
sitenumber(1,"site1").
sitenumber(5,"site5").
sitenumber(7,"site7").
sitenumber(8,"site8").
sitenumber(11,"site11").
sitenumber(15,"site15").
sitenumber(16,"site16").
siten(1,"site1").
siten(2,"site5").
siten(3,"site7").
siten(4,"site8").
siten(5,"site11").
siten(6,"site15").
siten(7,"site16").
sitename("site1","Cedar 1").
sitename("site5","Cedar 2").
sitename("site7","Cedar 3").
sitename("site8","Cedar 4").
sitename("site9","Lake Washington").
```

```
/*---------------------------------------------------------- portion of SITE1.DAT ----------------------*/
%**** the streamflow data
%
site1(1,1929,114.)
site1(1,1930,221.)
site1(1,1931,491.)
site1(1,1932,634.)
site1(1,1933,1025.)
site1(1,1934,1490.)
site1(1,1935,1241.)
site1(1,1936,871.)
site1(1,1937,178.)
site1(1,1938,655.)
site1(1,1939,978.)
site1(1,1940,339.)
site1(1,1941,361.)
site1(1,1942,296.)
site1(1,1943,484.)
site1(1,1944,296.)
site1(1,1945,1007.)
site1(1,1946,531.)
site1(1,1947,749.)
site1(1,1948,440.)
site1(1,1949,229.)
site1(1,1950,440.)
site1(1,1951,604.)
site1(1,1952,196.)
site1(1,1953,1508.)
site1(1,1954,500.)
site1(1,1955,380.)
site1(1,1956,339.)
site1(1,1957,220.)
site1(1,1958,629.)
site1(1,1959,1050.)
site1(1,1960,215.)
site1(1,1961,844.)
site1(1,1962,821.)
site1(1,1963,503.)
site1(1,1964,646.)
site1(1,1965,931.)
site1(1,1966,425.)
site1(1,1967,1121.)
site1(1,1968,885.)
site1(1,1969,533.)
site1(1,1970,723.)
site1(1,1971,950.)
site1(1,1972,737.)
site1(1,1973,557.)
site1(1,1974,1203.)
site1(1,1975,931.)
site1(1,1976,972.)
site1(2,1929,85.)
```

```
site1(2,1930,881.)
site1(2,1931,419.)
site1(2,1932,706.)
site1(2,1933,229.)
site1(2,1934,423.)
site1(2,1935,633.)
site1(2,1936,257.)
site1(2,1937,301.)
site1(2,1938,217.)
site1(2,1939,445.)
site1(2,1940,587.)
site1(2,1941,248.)
site1(2,1942,353.)
site1(2,1943,517.)
site1(2,1944,346.)
site1(2,1945,697.)
site1(2,1946,322.)
site1(2,1947,696.)
site1(2,1948,478.)
site1(2,1949,489.)
site1(2,1950,564.)
sitename("site11","Main stem of Tolt").
sitename("site15","North Fork of Tolt").
sitename("site16","South Fork of Tolt").
```