ED 201 304 IR 009 186

AUTHOR	Jarvis, John J.; And Others
TITLE	EZLP: An Interactive Computer Program for Solving Linear Programming Problems. Final Report.
INSTITUTION	Georgia Inst. of Tech., Atlanta.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	Sep 76
GRANT	NSF-SED-75-17476
NOTE	66p.
EDRS PRICE	MF01/PC03 Plus Postage.
DESCRIPTORS	Computer Assisted Instruction; Computer Science Education; *Linear Programing; *Man Machine Systems; *Models; *Problem Solving

ABSTRACT

Designed for student use in solving linear programming problems, the interactive computer program described (EZLP) permits the student to input the linear programming model in exactly the same manner in which it would be written on paper. This report includes a brief review of the development of EZLP; narrative descriptions of program features, including the driver, the editor, syntactical analysis, optimization and output, segmentation of EZLP, and machine dependence of EZLP; user instructions for entering, editing, and solving the model, as well as analyzing results, restarting and terminating the program, the HELP command, and external file-handling capabilities; and printouts of five terminal sessions providing simple and advanced examples of EZLP, as well as transportation, integer programming, and sensitivity analysis problems. A user's manual suitable for handout to students learning EZLP is appended. (CHC)

FINAL REPORT
NSF Grant Number SED75-17476

EZLP: An Interactive Computer Program for Solving Linear Programming Problems

John J. Ja[illegible], Project Director
V. Ed Unger, [illegible] Associate
Frank H. Cullen, Graduate Assistant
[illegible]s Papaconstadopoulos, Research Assistant

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

September, 1976

TABLE OF CONTENTS

ABSTRACT

This report discusses the development of an interactive computer program (EZLP) designed for student-oriented use in solving linear programming problems. The linear programming problem is inputted in the same way as it would be written on a sheet of paper.

The student may select either the (primal) simplex or the dual simplex method; the lower-upper bounded variables procedure with either method; and real (0.5) or rational (1/2) arithmetic for the calculations. EZLP has internal editing capability and is able to read from and write to permanent files.

During execution of EZLP, if the student has difficulty in remembering what to do next, he may utilize a HELP command to obtain general or specific information on EZLP's use.

Since EZLP was developed under a National Science Foundation grant a program listing on computer paper is freely available. A listing on cards or tape is available at a nominal charge to cover expenses. EZLP may not be resold.

Finally, a simplified user's manual suitable for handout to students learning EZLP for the first time is contained in Appendix A.

Chapter 1: INTRODUCTION

1.1 Background

Linear programming is a mathematical optimization technique utilized intensively in the fields of mathematics, management and engineering--especially within the areas of operations research and management science. The simplex method--the standard technique for solving linear programs--is an iterative method, and, as such, is dependent on the computer to carry out the rote calculations. Computerized simplex codes are available in small problem-solving versions and in large production versions in most university, private and governmental computing centers. Students find that interaction with these computer codes is often very difficult. The result is that small scale usage in the classroom or laboratory or for homework is discouraged. This bad experience carries over and tends to discourage the student's use of linear programming on the job.

In most universities today there are a number of undergraduate and graduate courses utilizing linear programming as a problem-solving tool. Georgia Tech is typical with as many as 6-10 undergraduate courses and 15-20 graduate courses utilizing linear programming. In these courses the student is expected to develop modelling skills with linear programming, appreciate the simplex method (or its variants) as a solution tool and to interpret the output results of the simplex method in the particular problem situation. In addition to course work there are usually a number of students performing research on new techniques which require a check by solving small linear programming problems. Hand calculations are good up to about four constraints and ten variables. Beyond this number the student must go to the computer in order to have the calculations performed, otherwise, negative feedback results. The student remembers the difficult[illegible] with solving [illegible] model and not the pow[illegible] of the modelling technique. When the [illegible] gets to the computer he usually finds a [illegible] new set of [illegible]ies with regard to the fact that in order to [illegible] with his model he must learn a computer-oriented [illegible] -oriented) "input language".

A number of institutions [illegible] Northwestern University, Stanford University, the University [illegible] Texas, and Georgia Tech have developed linear programming [illegible] programs which are more directly student oriented in their design.

In the School of Industrial [illegible] Systems Engineering at Georgia Tech an experimental project has been underway since 1971 to develop a student-oriented conversational linear programming code (EZLP) for use on small problems (up to 50 constraints and [illegible]0 variables). The innovation is in the special way that the student interacts with the computer. The student inputs his linear programming model to the computer in exactly the same manner that he would write it down on paper. Evolution of this code has been through class projects. Despite limitations in its initial design, EZLP was nevertheless widely accepted by the students.

1.2 Initial Project Effort

The National Science Foundation agreed to support a concerted effort, during the [illegible]75-1976 academic year, to redesign and develop a conversational computer code based on the concepts developed at Georgia Tech.

To accomplish the given task, a Georgia Tech team was organized. The Georgia Tech team consisted of:

Dr. John J. Jarvis, Project Director

Frank H. Cullen, Research Assistant

Chris Papaconstadopoulos, Research Assistant

The team had the overall responsibility of designing, developing, coding and documenting the computer system.

An Advisory Committee of eminent researchers and practitioners in the field of linear programming was also organized to provide guidance to the Georgia Tech team. [illegible] Advisory Committee consisted of the following individuals:

1. Dr. Claude [illegible]
   Northwestern University
2. Dr. Harvey [illegible] [illegible]nberg
   Federal Energy Administration
3. Dr. Marvin [illegible]. Griffin
   University of Alabama
4. Dr. Michael E. Thomas
   University of Florida

This committee had significant impact on the design of the final computer system.

1.3 Initial Project Plan

During the early stages of the project the Georgia Tech team devoted a great deal of time and effort to the development and documentation of a preliminary design of a computer system for EZLP. This documentation was submitted to the Advisory Committee for their review.

A meeting of the Advisory Committee was conducted on the Georgia Tech campus to discuss the preliminary design. This discussion lead to a much improved design. On advice of the committee, the fundamental concept of complete machine independence was replaced by a moderate level of machine dependence to permit substantial reduction in program size and complexity. Also, a number of general features were eliminated from the initial design while many others were added. The Georgia Tech team was very pleased with the structure that emerged from that meeting.

The Advisory Committee also suggested, at its meeting, that the Georgia Tech team retain a certain degree of flexibility in its development of [illegible] so that the Tech team might be able to react in an expeditious manner [illegible] potential difficult situations which could arise during coding.

The remainder of the initial grant period, following the Advisory Committee meeting, was spent developing and testing the EZLP code. With a few changes, the initial EZLP system developed was the one which emerged from the Advisory Committee meeting.

1.4 Follow-on Project Effort

The initial project phase concluded with the development of the basic EZLP system. The project then proceeded into a second phase consisting of dissemination, formative evaluation, and modification/improvement.

During the second phase, the dissemination task consisted of mailing literature on the basic EZLP system to approximately one hundred and fifty (150) departments of Industrial (and Systems) Engineering, Operations Research, Management Science, Computer Science and Mathematics throughout the United States. The literature described the EZLP system, its functions and operation. Copies of the computer code were offered at nominal or no charge to interested institutions. The objectives of this offering were to (1) achieve wide dissemination of the project results, and (2) generate a subset of implementing institutions from which a formative evaluation of the EZLP system could be conducted.

Some two dozen institutions requested the EZLP code for implementation. It was anticipated that a group of these institutions, providing a cross section of EZLP users, would be identified for participation in a seminar on EZLP to be held at Georgia Tech. However, it soon became evident that the critical link would not be understanding the EZLP system, but, instead, would be getting EZLP operational on the myriad of different computers (UNIVAC, Burroughs, Prime, IBM, etc.) involved.

After discussions with NSF, it was decided to carry the EZLP seminar to the users, instead of having the users come to Georgia Tech. A qualified graduate research assistant was sent to several targeted institutions to (1) solve EZLP implementation problems, (2) give a seminar/demonstration of the EZLP system and its capabilities, and (3) establish the necessary mechanism for receiving comments and evaluations of the EZLP system. Several institutions were targeted on the basis of their interest and enthusiasm and the type of computer system involved. It was hoped that the formative evaluation could include results of experience on as many different computer systems as possible.

The implementation, testing and evaluation process proved invaluable to the continued improvement of the EZLP system. EZLP users evaluations included (1) modifications of the code to provide easier installation of differing computer systems, (2) changes in the materials, handouts, etc. describing EZLP and its uses, (3) expansion and improvement of the EZLP code to include additional features requested by the users, (4) modularization of the EZLP system to overcome the problems associated with core memory requirements for operation of the system, (5) changes in EZLP command structures and output formats to provide more intuitive understanding of the system operation and results, and (6) the alleviation of minor bugs and difficulties associated with the EZLP system.

The EZLP system which resulted from this testing and evaluation process is greatly improved. EZLP is currently operational on approximately two dozen computer installations throughout the United States. This list includes academic institutions, governmental agencies and private firms. EZLP users report great popularity and success of EZLP. The Georgia Tech team is pleased with the resulting system and its acceptance by others. EZLP is becoming known worldwide, and already copies of the code have been sent to several European countries.

The remaining chapters discuss the specific structure of EZLP and its use. Appendix A contains a simplified user's manual which could be passed out to students learning EZLP for the first time.

1.5 Machine Dependence of the EZLP System

The EZLP computer system has been intentionally designed to minimize the level of machine dependence within the constraint of reasonable program size and complexity. EZLP has been coded in FORTRAN and special forms which may be available only on the CDC Cyber 74 (the machine used) are avoided. Those places in the code where machine dependence was unavoidable are few in number and clearly identified.

Section 2.8 of Chapter 2 discusses the required changes to convert the machine dependent portions of EZLP to other computer systems.

1.6 Availability of the EZLP System

Development of the EZLP computer system was sponsored by a grant from the National Science Foundation. As such, the EZLP computer system is freely available to anyone desiring a copy. A computer listing on paper is available at no charge. There is a nominal charge, to cover expenses, for a listing on cards or tape. EZLP may not be resold. For further infor-

write

Dr. John J. Jarvis, EZLP Project Director
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

1.7 Continuing Effort

EZLP is an evolving computer system. Effort continues at Georgia Tech to maintain and improve the EZLP code. We invite comments and suggestions on improving EZLP.

Chapter 2: EZLP - A NARRATIVE DESCRIPTION

2.1 Introduction

This chapter presents the workings of EZLP in a narrative form. Emphasis is given to the identification of the components of EZLP, their attributes, and a discussion of their mutual interaction. Coded in standard FORTRAN, EZLP nonetheless has certain functions which cannot be written in machine-independent code. These are discussed in the last part of this chapter.

2.2 A General Outline

In overview, EZLP consists of three main subprograms tied together by a controlling main program, or "driver." The three main subprograms are:

a. Editor - the ability to create and maintain (update) a current model kept physically in an exterior mass storage device

b. Syntactical Analysis - inputting the model statements from the external file created by the EZLP Editor, performing parsing and syntax-checking operations, and generating a tableau for input to the optimization process

c. Optimization - a solution and output of a model in tableau form by simplex procedures

Figure 2.1 presents a general diagram of the program flow of EZLP which also indicates the interaction of the three main subprograms.

2.3 The Driver

The main program, or driver program, has several functions. Among these are:

a. distribute and pass control among the three main subprograms

b. perform ancillary operations not handled by the subprograms

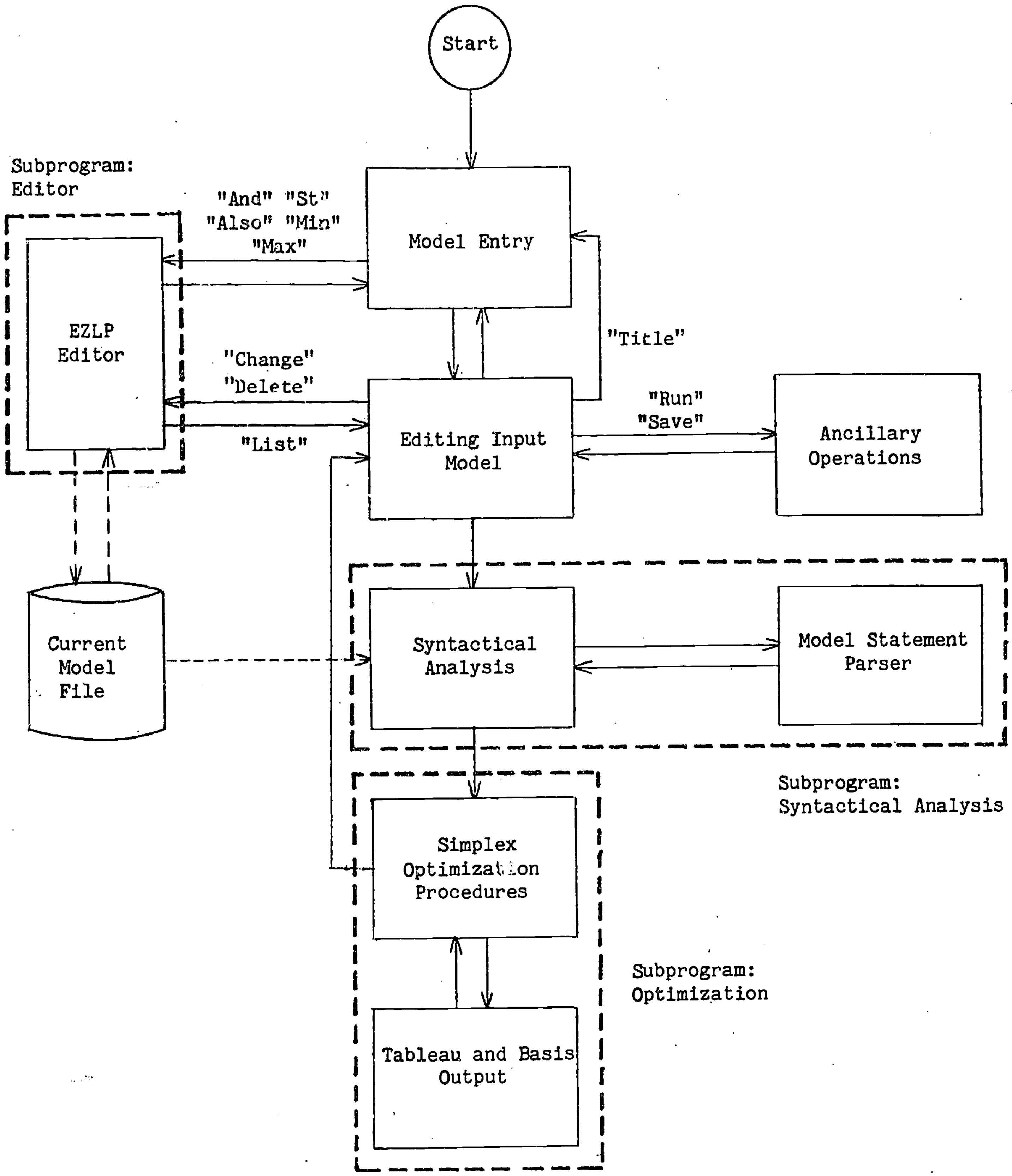


Figure 2.1: EZLP Program Logic Flow

Upon program execution, the driver program assumes control and examines the first input statement. If the keyword (the first word) of the input statement is an editor associated keyword (e.g., AND, MIN, CHANGE), the driver continues to parse the input statement to determine the row name if it is there and to construct the default value if it is not; control is then passed to the editor subprogram. If the keyword of the input statement is associated with an ancillary operation (e.g., RUN, SAVE, or PRINT), the driver continues to parse the input statement to determine what internal program switches are to be set, and in the case of the SAVE statement, perform the requested file operations. If the keyword is USE, to trigger a solution attempt, the driver parses the remainder of the input statement to determine the simplex method to be used, passes control first to the syntactical analysis subprogram, and then to a tableau-building subroutine and the optimization subprogram in succession, if there are no syntactical errors in the model.

In Figure 2.1 the driver program may be considered to be everything not within a subprogram boundary.

2.4 The Editor

The editor subprogram is primarily responsible for model file maintenance and performs four principal functions:

a. Adds a statement to the model file

b. Deletes a statement in the model file

c. Changes a statement in the model file

d. Lists either a single row or the entire contents of the model file

Model File Organization

The model file itself is a simple linked-list structure with chaining used for continuation lines. Two vectors in the COMMON block EDIT, CNTNME and LINKS, store the alpha name and the relative location of the first record for the particular row of the model on mass storage. This record has a forward link to a continuation record if used. This forward link is zero (0) if there are no further continuations after the current record. The program logic and passed parameters to the mass storage read and write subroutine are such that first-available and other random access techniques can be used on computers of various manufacture.

Subprogram Operation

Upon passage of control and the appropriate parameters, the Editor accesses the model file to perform the desired function, and returns control to the driver which then reads in the next input statement.

Continuations of rows in the model are handled directly by the Editor, which reads in successive input statements and sets up the chaining relationship until the continuation situation no longer exists.

2.5 Syntactical Analysis

The syntactical analysis subprogram is the process wherein the model file contents are read in, parsed, checked for errors in syntax, and translated into a coded form which is subsequently transformed into a tableau structure.

In overview, the syntactical analysis subprogram has two main entities:

a. the model statement parser, which takes lines from the model file and generates a uniform symbol table in which the components of the model line (e.g. coefficient, name or identifier, delimiter) are described and;

b. the analysis routine which uses as input the uniform symbol table, checks for syntax errors, and outputs a quasi-diagonal tableau which later serves as input to a tableau-building subroutine which adds slack and artificial variables and generates a rectangular tableau structure.

Other output from the syntactical analysis subprogram includes:

a. a vector containing row names as they appear in the tableau
b. a vector containing variable names as they were first mentioned in the model file
c. vectors containing lower and upper bounds for each variable in the same order as (b)
d. A designator of which rows of the tableau are associated with objective functions
e. an indication of which objective function row is to be optimized during phase two of the optimization process

In the instance that there are syntax errors present in the model file, descriptive error messages and pointers are printed in an attempt to pinpoint the exact location of the trouble.

The memory locations used by the syntactical analysis are focused in the COMMON area OPT and WORK.

2.6 Optimization and Output

The optimization and output subprogram is the process by which the simplex operations are performed on the rectangular tableau created by the tableau-building subroutine which, in turn, receives its input from the syntactical analysis subprogram. Depending upon the value of the parametric switch METHOD set by the driver program during the parsing of the USE statement, the optimization process selects from four basic algorithms: primal or dual simplex with or without rational arithmetic. The lower-upper bounded simplex approach is used throughout the optimization routines. The different results obtained by the specification of the UPPER parameter in the USE statement owe to the generation of explicit bound constraints by the syntactical analysis subprogram when UPPER is not specified.

The primal algorithms (real and rational) employ a two-phase method for which the Phase 1 objective function and the starting basis are determined by the tableau-building subroutine. At the successful completion of Phase 1, the Phase 2 objective function (which has been stored as a null constraint in the tableau) is written over the Phase 1 objective function and the process resumes.

The dual algorithms begin with the creation of the Phase 2 objective function from the appropriate null constraint (objective function) in the tableau.

The rational algorithms require the maintenance of both a numerator and a denominator for each tableau entry (and each element of the right-hand side). For example, the numerator for an element in an array may be stored in position I, while the corresponding denominator will be stored in position J+I, where the Jth position of the array held the value of the numerator of highest subscript index.

At the end of each simplex iteration, or both before and after the optimization process has been completed, there might be output specified by the user via a PRINT statement. In this instance, control within the optimization process is passed to a point which prints out the values of the nonzero variables, the current basis and basis activity, or the current tableau and updated objective function. Control is then returned to the appropriate place in the optimization process.

Upon reaching an optimal solution, control within the optimization process passes to a point at which the default output (the optimal primal and dual solutions) is printed out. Even if no other output is specified or requested, the default output is printed.

When the printing default output is completed, control returns to the driver program which then reads in and parses the next input statement.

The memory locations used by the optimization and output subprogram are focused in the COMMON area OPT and WORK.

2.7 Segmentation of EZLP

In dealing with computers with smaller capacity than large-scale university computing systems, memory economy via code segmentation or overlay is often of considerable interest.

The modular subprogram concept allows some or all of the subprogram code to be overlayable, and hence less wasteful.

Specifically, the three subprograms are mutually overlayable as well as those functions which serve only one of the subprograms. Graphically, we can represent the code components of EZLP as a tree, the root of which is non-overlayable, and the separate branches of which form the overlays. In Figure 2.2, this tree structure is represented in terms of the actual FORTRAN subroutines and function names.

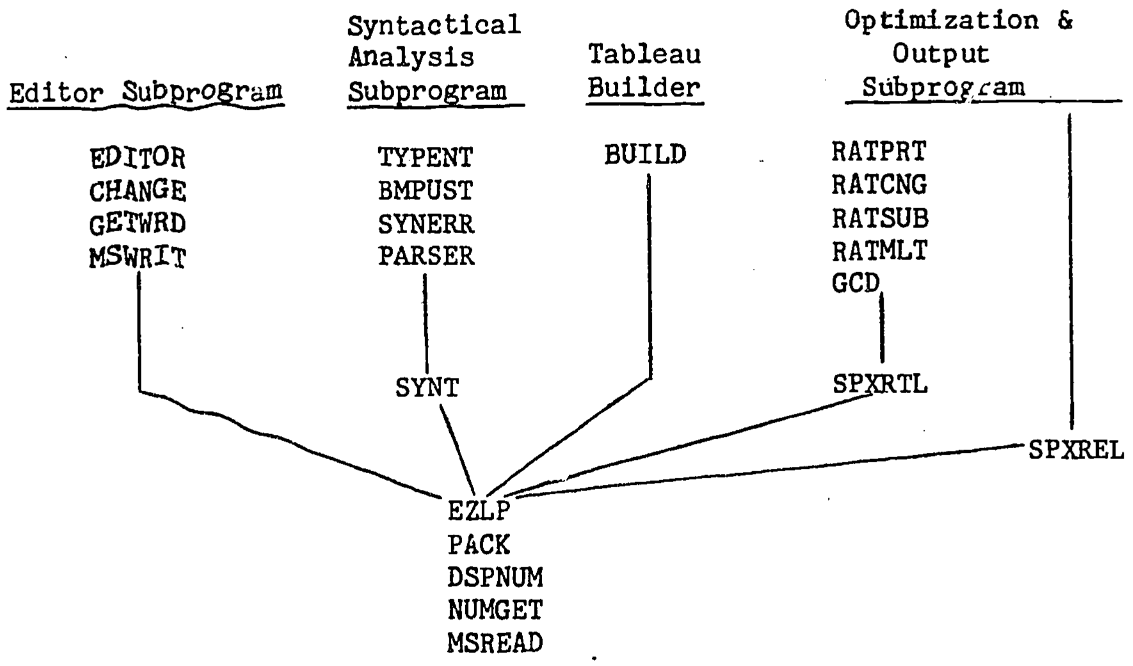


Figure 2.2: Segmentation Tree-Structure of EZLP

2.8 Machine Dependence of EZLP

There are two aspects of the code of EZLP which are likely to cause difficulties when an attempt is made to convert the code to a machine other than the one for which EZLP was written (the CDC Cyber 74). These are

a. Alpha variables - packing and unpacking, and manipulation

b. File manipulations - particularly with respect to error and end-of-file conditions

Alpha Variables

The storing of alphanumeric information in variables has always been a weakness in the design of FORTRAN. Consequently, the use of either integer or real variables to store alphanumeric information is bound to be non-standard and change from machine to machine. For the purpose of program design and implementation, EZLP uniformly uses real variables. For those compilers and computers which require that some other type be used for alpha variables, explicit type statements must be added to those portions of the code affected for the alpha variables.

A basic assumption of the EZLP code is that the alpha variables can hold at least eight (8) characters and are appropriately filled so that a comparison with a Hollerith constant is meaningful. Hexadecimal machines (or character machines) normally have type statements which allow the length of the variable in bytes (characters) to be specified. Where applicable, these constructs should be used.

An often-used subroutine in EZLP for composing a single eight-character alpha variable from eight single-character alpha variables is called PACK.

This subroutine is totally machine-dependent and must be changed for different computers.

File Manipulation

EZLP employs a number of files during execution. These files are summarized in Table 2.1. These files, with the possible exception of file are strictly sequential files and require no special programmatic conside 3. File 8, the model file, because of the random way in which editing is applied to it, can be effectively used as a random access file. Since random access is normally a machine-dependent function, EZLP separates out the random access read and write functions into separate subroutines MSREAD and MSWRIT. These two subroutines must be re-written for different computers.

Table 2.1: EZLP Files and Their Description

File Number	Description
5	Input file (from the terminal)
6	Output file (to the terminal)
7	Message file (for syntax error messages)
8	Model file (linked and chained mass storage)
9	Run file (alternate input file)
10	Save file (model output file)
11	Help stored-text file (instructions)

End-of-file checking is also achieved differently with different FORTRAN compilers. On the CYBER 74, the test is the implicit function EOF(u). Occurrences of this function should be replaced by the appropriate READ statement construct for the local FORTRAN compiler. For example, the sequence

```
READ (INFILE, 102) (BUFFER(I), I=1, 80)

  IF (EOF(INFILE) .NE.0) GO TO 320
```

could be replaced by something resembling

```
      READ (INFILE, 102, END=320) (BUFFER(I), I=1, 80)
```

if such was the appropriate end-of-file construct.

Chapter 3: USER INSTRUCTIONS

3.1 Introduction and Notation

This chapter is designed to describe the operation of EZLP from the user's point-of-view. The material covered here is somewhat more comprehensive than that which would be required for the user with simple problems. Appendix A contains a simplified user's manual.

For clarity of presentation, this chapter uses BNF notation in describing syntactical rules. Briefly, capitalized words indicate keywords, [] indicates an optional clause, $\{\ \}$ indicates a choice of two or more alternatives, and <> enclose a descriptive term for a syntactical entry.

3.2 Starting EZLP

The exact syntax of executing EZLP will vary from computer to computer and from installation to installation, and so cannot be explicitly stated here. For the sake of clarity, let it be assumed that EZLP has been executed and is awaiting input from the user.

At this point, the user has the option of proceeding to enter the model (3.3) or getting a brief (about two pages) description of how EZLP is operated. This description, in the absence of this chapter or other documentation should supply a minimum of information to the user so that he can use the program. To obtain this brief description, the user must enter the following command:

```
HELP BRIEF
```

This command is discussed further in section 3.7.

3.3 Entering the Model

The first thing the user must do is enter the model. The entry of the model follows certain rules - there are rules concerning the order in which inputted statements must be organized, and there are rules which govern the way in which different types of statements are made up. Before these rules are discussed, it would be best if some basic ideas were explained: A statement is a single logical input. A constraint is a statement. The objective function is a statement. A delimiter is a single character which serves to terminate one entry and perhaps begin the next. For the entry of the model, the space character (␢) is the usual delimiter. A name provides the convenience of letting the user assign unique identifiers to variables, constraints, and objective functions. The names themselves are composed of from one to eight characters. The first character must be alphabetic (A-Z), while the remaining characters must be alphabetic, numeric (0-9), or numerics separated by commas. X2, PROFIT, ITEM3, and Y2,3 are examples of acceptable names, while ITEM-3 and $VAR are not. A space character cannot be part of a name. EZLP reserves certain names for its internal use. Row names of the form ROW#n are reserved for each objective function or constraint statement which was unnamed by the user. This generated name can be used for editing purposes. In addition, EZLP uses the names SLK#n and ART#n for the slack and artificial variables that it generates. Since the user may not input any name containing the # character there can never be any confusion between user-generated names and EZLP-generated names. An input-line corresponds to a line of input on the terminal. In batch processing, this is equivalent to a card image.

If desired the user may indicate that heading information is to be printed at the top of the optimal solution output. The user must type in

```
TITLE <heading information>
```

where <heading information> appears as a heading at the top of the output.

The actual entry of the model is divided into two basic parts: the objective function and the constraints.

Entering the Objective Function

The syntactical rule for the entry of the objective function is:

$$\left\{\begin{matrix}\text{MIN}\\ \text{MAX}\end{matrix}\right\} \text{ or } \left\{\begin{matrix}\text{MINIMIZE}\\ \text{MAXIMIZE}\end{matrix}\right\} [\text{<objective-function-name>}]\text{:} \quad \text{<arithmetic expression>}$$

where <objective-function-name> is an optional user-assigned name for the objective function and <arithmetic expression> is a linear combination of variable-names.

Note that the colon (:) is required at all times.

If the objective function statement is too large to be entered on one input line, continuation of the statement is accomplished by placing an ampersand (&), after the last character of the current input line and continuing the statement on the next input line. There are no limits on the number of continuations allowed for a single statement.

Examples of possible objective function statements:

1. MIN: 2 + 3X2 - 2.3X4
2. MAX PROFIT: 3INCOME - 4EXPNSE - .673OVERHD
3. MIN COST: 2.4VAR1 - 3.53VAR2 + .004VAR3 &

   -VAR4

Alternate Objective Functions

In some applications of linear programming, it is of interest to consider a number of possible objective functions subject to the same set of constraints. EZLP allows for the entry and subsequent optimization of the entered model in coordination with alternate objective functions.

The syntactical rule for the entry of an alternate objective function is:

ALSO [<objective-function-name>]: <arithmetic expression>

Notice that the above differs from the ordinary entry of the objective function in that the keyword "ALSO" is required and replaces the keywords "MIN" and "MAX".

Each model must contain exactly one primary objective function (using the keywords "MAX" or "MIN"). However, a model may contain any number of alternate objective functions. EZLP operates with only one objective function at a time.

Specifying an alternate objective function for the purpose of optimization is accomplished just prior to specifying the method of solution. This is discussed in Section 3.5 ii.

Entering the Constraints

The constraints associated with normal linear programming model can be broken down into three general classes:

1. Simple arithmetic constraints - those constraints which involve more than one variable and only one relational operator; and
2. Range constraints - those constraints which involve more than one variable and two identical operators; and
3. List constraints - those constraints which involve only one variable. Constraints of this type usually specify upper or lower bounds on a particular variable or a list of variables.

Simple Arithmetic Constraints

The syntactical rule for the entry of a simple arithmetic constraint is:

$$\left\{\begin{matrix}\text{ST}\\ \text{AND}\end{matrix}\right\}$$ [<constraint-name>]: <arithmetic expression>

<relational operator> <arithmetic expression>

where <constraint-name> is an optional user-assigned name for the particular constraint. The <arithmetic expression> is a linear combination of variable names as described in the objective function statement above; and <relational operator> takes one of the following forms:

1. {=> or >=} (greater than or equal to)
2. = (equal to)
3. {=< or <=} (less than or equal to).

Note that the colon is required at all times.

If any particular constraint statement is too large to be entered on one output line continuation of the statement is accomplished by placing an ampersand (&) after the last character of the current input line and continuing the statement on the next input line. There are no limits on the number of continuations allowed for a single constraint statement.

The first constraint statement should begin with the keyword "ST", which stands for "subject to". Subsequent constraint statements must begin with the keyword "AND".

Constraint statements which include the optional <constraint-name> are easily referenced for editing by this user-assigned name. Unnamed constraints are assigned a name by EZLP so that the user can also reference and edit these constraints. This is further discussed in section 3.4.

Examples of possible simple arithmetic constraints:

1. AND CNSTR2: 20WIDTH2 <= 15 - LENGTH
2. AND POWER: TOTAL + 3TIMEAVL> = 16.45
3. AND: 4.3X1 + 6X7 = 4.57

Range Constraints

EZLP is designed to accommodate range constraints. The syntactical rule for the entry of range constraints is:

```
{ST }
{AND} [<constraint-name>]: <constant-1><inequality-operator-1>
       <arithmetic expression> <inequality-operator-2><constant-2>
```

where the following rules apply:

1. <inequality-operator-1> and <inequality-operator-2> must be exactly alike and must be either >= or <=.
2. If the relational operators are >=, then <constant-1> must be greater than or equal to <constant-2>
3. And, if the relational operators are <=, then <constant-1> must be less than or equal to <constant-2>.

Examples of possible range constraints are:

1. AND: 4 <= 3x1 + 4.2x3 - 4x4 <= 6.2
2. AND WORKERS: 500 >= 6.3FORCE1 - 5.3FORCE2 >= 243
3. AND CONSTRNT: 3.4 <= 7+ VAR3 =<8.46

List Constraints

The syntactical rule for the entry of a list constraint is:

```
AND [<constraint-name>]: {ALL [OTHER] VARS} {<relational operator><constant>}
                         {<variable-list>  } {            URS             }
```

where <constraint-name> and <relational operator> are described above for simple arithmetic constraints; and <variable-list> is a list of one or more variable names, separated by commas (,).

The reserved phrase "ALL VARS" indicates that the bound specified applies to all variables in the model. The reserved phrase "ALL OTHER VARS" indicates that the bound specified applies to all variables which are not included in another list constraint for the same type bound (i.e., upper or lower).

The reserved word "URS" stands for "unrestricted in sign".

The default option for all variables if they do not appear in a list constraint is "URS". EZLP prints a notification of the unrestricted variables.

Examples of possible list constraints:

```
1.  AND:  X1, X2, X3 <= 0
2.  AND:  ALL VARS >= 0
3.  AND:  OIL <= 375.43
4.  AND:  ALL OTHER VARS <= 1
5.  AND:  PROFIT URS
6.  AND MYCNSTR:  WIDTH,LENGTH >= 0
```

An example of an entry of a complete model:

```
MIN:   10.5WIDTH1 + 11.8WIDTH2 - 30LENGTH
ST CNSTR1:  15LENGTH - 2WIDTH1 <= 10
AND CNSTR2:  20WIDTH2 + LENGTH < = 15
AND:  ALL VARS >= 0
```

3.4 Editing the Model

After completion of the model entry, the user has the ability to perform certain editing functions. These functions are:

1. Adding a constraint or objective function
2. Deleting a constraint or objective function
3. Changing a constraint or objective function
4. Listing the model in whole or in part

i) Adding a Constraint or Objective Function

At all times the user has direct access to his model. Thus, addition of a new constraint or objective function to the end of the current model may be accomplished by simply typing the appropriate constraint or objective function statements. EZLP will automatically append the new statement to the previous model.

For example, suppose that the user has attempted to solve his model and this attempt resulted in an unboundedness indication. Having determined that he failed to require nonnegativity he may do so by simply typing

```
AND:  ALL VARS >= 0
```

If one wishes to insert a constraint into the middle of the current model, this may be accomplished by typing

$$\text{INSERT}\left\{\begin{matrix}\text{AFTER}\\ \text{BEFORE}\end{matrix}\right\}\langle\text{row-name}\rangle\ \{\text{model entry statement}\}$$

where <model entry statement> is either a constraint or objective function statement. Examples of the INSERT command are:

```
1.  INSERT AFTER ROW#2 AND:  X1 + 2X2  <= 7
2.  INSERT BEFORE CONST6 AND CONST5:  7  <= X1  <= 9
3.  INSERT AFTER ROW#3 ALSO:  3X1 + 2POWER
```

ii) Deleting a Constraint or Objective Function

The syntactical rule for the deletion of a constraint is:

```
DELETE <row-name>
```

where <row-name> is either the user assigned name, or in the absence of this name, the row name assigned by EZLP. The assigned name is simply ROW#n, where n is the number of the model entry as it was entered. Examples of assigned row-names are ROW#4, ROW#13, and ROW#123.

1. DELETE ROW#13
2. DELETE SURPLS

iii) Changing a Constraint or Objective Function

The syntactical rule for the changing of a constraint or the objective function is:

```
CHANGE <row-name> "<string1>"<string2>"
```

where <row-name> must be identical to some existing row name.

The CHANGE command will replace <string1> by <string2>. The construct "<string1>"" will delete <string1>.

Examples of possible CHANGE statements are:

1. CHANGE CONSTR1 "ENG"G"
2. CHANGE SURPLS "3.2""

iv) Listing the Model

The syntactical rule for listing the model on the terminal is:

```
LIST [<row-name>]
```

where the optional <row-name> is included if only a single row is to be printed. If <row-name> is omitted the entire model will be printed. Examples of possible LIST commands are:

1. LIST
2. LIST ROW#5
3. LIST MYCSTR

3.5 Solving the Model

Once the model has been entered and edited to the user's satisfaction, the user takes the following steps:

1. Specifying the Desired Output (this step is optional and, if omitted, results in only the optimal solution being printed).
2. Specifying the Objective Function to be Optimized (this step is optional and is only to be used when selecting an alternate objective function).
3. Specifying the Method of Solution.

i) Specifying the Desired Output

If the user wishes to obtain more output than the optimal solution he may use the PRINT command.

The syntactical rules for the PRINT command are:

$$\text{PRINT [INITIAL] [FINAL]} \left\{ \begin{array}{l} \text{VARS} \\ \text{BASIS} \\ \text{TABLEAU} \\ \text{ALL} \\ \text{NONE} \end{array} \right\} \text{[,<frequency count>]}$$

This statement concerns the output on the terminal of information directly related to the optimization process. The following descriptions apply:

VARS:	prints the name and value of each nonzero primal variable; and the name and value of each nonzero dual variable.
BASIS:	prints the names of the variables in the basis and the objective function value.
TABLEAU:	prints the tableau.
ALL:	prints all of the above.
NONE:	prints only the optimal solutoin.

<frequency-count>--the PRINT statement is executed every <frequency-count> iterations. If this optional clause is omitted, the PRINT statement will be executed after every iteration unless the keywords INITIAL and/or FINAL are used. When ALL or NONE are present <frequency-count> is ignored.

INITIAL:	prints the requested information only for the initial iteration.
FINAL:	prints the requested information only for the final iteration. (Note that INITIAL and FINAL may be used together.)

Examples of possible PRINT commands are:

1. PRINT VARS 5
2. PRINT ALL
3. PRINT BASIS, VARS,3
4. PRINT FINAL TABLEAU
5. PRINT INITIAL, FINAL BASIS

ii) Specifying the Objective Function to be Optimized

In the event that the user has entered alternate objective functions, he may wish to specify the name of the objective function to be optimized. If this is not done, EZLP defaults to either the original objective function entered with the model or the last objective function name specified in a prior ALTOBJ statement. The syntactical rule for the specification of the objective function command is

$$\text{ALTOBJ} \begin{Bmatrix} \text{MIN} \\ \text{MAX} \end{Bmatrix} \langle\text{objective-function-name}\rangle$$

If an ALTOBJ command has been previously given, then to determine the row name of the current objective function being used, the user should type

```
ALTOBJ   STATUS
```

iii) Specifying the Method of Solution

The specification of the method of solution triggers an attempt by EZLP to solve the entered model and produce output as specified in 3.5.i.

The syntactical rule for specifying the method of solution is:

$$\text{USE} \begin{bmatrix} \text{RATIONAL} \end{bmatrix} \begin{bmatrix} \text{UPPER} \end{bmatrix} \begin{Bmatrix} \text{PRIMAL} \\ \text{DUAL} \end{Bmatrix}$$

Specifying RATIONAL keeps all data in rational form, i.e. "0.5" would become "1/2". Since the RATIONAL option requires storing both a numerator and a denominator matrix, the maximal allowable problem size is cut in half under this option. Further, all coefficients and constants must be entered as integers.

Specifying UPPER before PRIMAL or DUAL will cause the lower-upper bounded primal or dual algorithm to be used.

Examples of possible USE commands are:

1. USE PRIMAL
2. USE UPPER DUAL
3. USE RATIONAL PRIMAL
4. USE RATIONAL UPPER PRIMAL

3.6 Sensitivity Analysis of the Results

Once an EZLP model has been optimally solved by any of the USE commands, the user has the option of requesting a sensitivity analysis of the cost coefficients or the right-hand-side constants. The syntax of this command is:

$$\text{RANGE} \left\{ \begin{array}{c} \text{RHS} \\ \text{OBJ or OBJFCN} \\ \text{ALL} \end{array} \right\} \left[\left\{ \begin{array}{c} < \text{name} > \\ \\ \text{ALL} \end{array} \right\} \right]$$

The effect of the RANGE command is the determination of lower and upper limits on various model parameters which maintain the current (optimal) basis. Examples of possible RANGE commands and their effects are:

	COMMAND	EFFECT
1.	RANGE RHS ROW#1	Provides lower and upper sensitivity limits for the right-hand-side constant for the ROW#1 constraint.
2.	RANGE OBJ X22	Provides sensitivity information for the cost coefficient for variable X22 in the objective.
3.	RANGE OBJ, ALL	Provides sensitivity information for all objective function coefficients.
4.	RANGE ALL	Provides sensitivity information for all objective function coefficients and all right-hand-side constants.

If a constraint specified in the RANGE command is, itself, a range constraint, then separate sensitivity information will be printed for the lower constant and the upper constant of the constraint.

3.7 Restarting and Terminating EZLP

At the completion of any model solution the user has three options:

1. Editing the current model and re-solving it,
2. Starting fresh with a new model, or
3. Terminating EZLP

The first of these options is accomplished by simply entering the appropriate edit statements (see 3.4) followed by the appropriate USE commands (see 3.5).

The second of these options is accomplished by the RESTART command. The syntax of this command is:

```
RESTART
```

The RESTART command re-initializes all areas. The effect of this command is to clear out the current model and print options.

Terminating EZLP is accomplished by the END command. The syntax of this command is:

```
END
```

3.8 The HELP Command

This command allows the user, after he has started the program, to obtain a short set of instructions on how to solve simple LP problems. The user, at any point in his run, may request additional help in the following manner:

```
HELP [<keyword>]
```

where the optional clause keyword refers to abbreviated instructions concerning a particular area of interest. A list of keywords is:

General Keywords:

BRIEF	(briefly describes the general features of EZLP)
EDIT	(describes edit commands)
EXAMPLES	(presents examples of EZLP models)
FILES	(discusses external file handling capabilities)
KEYWORDS	(gives the current list of keywords)
SOLVING	(indicates the methods and options for optimization and discusses output options)
MODEL	(discusses model entry syntax)

Specific Keywords:

ADD	(indicates how to add a constraint or objective)
ADVANCED	(presents an advanced example)
ALTOBJ	(indicates how to specify an alternate objective function for optimization)
CHANGE	(indicates how to change a constraint or objective)
CONT	(discusses the continuation of model statements)
CONS	(describes the options for inputting constraints)
DELETE	(indicates how to delete a constraint or objective)
LIST	(discusses the list options for model display)
NAMES	(provides a definition of acceptable variable names)
OBJ	(discusses objective functions)
PRINT	(discusses the output print options)
RUN	(describes procedure for inputting a model from an external file)
SAVE	(discusses procedure for saving a model)
TITLE	(discusses the options for method of optimization)
USE	(provides a definition of acceptable variable names)

If the keyword is omitted, EZLP will print a short description of how to obtain additional information on the operation of EZLP.

3.9 External File-Handling Capabilities

There are certain situations wherein the user wishes to eliminate input effort by storing all or part of his input on a mass storage device. EZLP provides for this capability with two commands: the RUN command and the SAVE command.

i) The RUN Command

At any point in the execution of EZLP, the user may elect to refer EZLP to a mass storage file for subsequent input. This input must be in the form of 80 character source records and must be accessible sequentially by EZLP. The syntax of the RUN command is

```
RUN <file-name>
```

If the RUN command is inputted, EZLP will read the model directly from the working file <file-name>, print a question mark and await the next command.

ii) The SAVE Command

Upon completion of model entry, the user may elect to save the input model in source form in a mass storage file. The syntax of the SAVE command is

```
SAVE <file-name>
```

If the SAVE command is inputted, EZLP will write the current model into the working file <file-name>, print a question mark and await the next command.

Chapter 4: TERMINAL SESSIONS

4.1 Introduction

This chapter contains several examples of EZLP used to solve normal linear programming problems. These examples include:

1. Simple example -	getting on, solving a simple problem, and getting off.
2. Advanced example -	naming rows, editing the model, requesting additional output, and responding to an error in syntax.
3. Transportation problem example -	two sources and two sinks to illustrate multiply-indexed variable names and the treatment of primal redundancy.
4. Branch and Bound example -	assignment of titles to optimal solution output in addition to using the editing features of EZLP to easily solve a small integer programming problem using the Branch and Bound method.
5. Sensitivity Analysis -	ranging applied to all constraints and all variables of a simple example.

4.2 Session 1: A Simple Example of EZLP

```
 EZLP - VERSION 9/17/76
 TYPE HELP IF YOU HAVE QUESTIONS, OTHERWISE PROCEED
? MAX: 3X1+2X2
? ST: 5X1+3X2 <= 19
? AND: X1-X2 >= 3
? AND: ALL VARS >= 0
? USE RATIONAL PRIMAL

       ** NO UNRESTRICTED VARIABLES IN THE MODEL **

                 **  VARIABLE LIST  **


  X1         X2



              S O L U T I O N
              ---------------

   OBJECT              MAXIMIZE  ROW#1
   Z                        23/2
   ITERATIONS             2

  V A R I A B L E   S E C T I O N
  -------------------------------
 NAME      ACTIVITY LEVEL     LOWER BOUND     UPPER BOUND     REDUCED COST

 X1                 7/2            0             POS INF          0

 X2                 1/2            0             POS INF          0


   C O N S T R A I N T   S E C T I O N
   -----------------------------------
 NAME      SLACK ACTIVITY           DUAL PRICE        RHS VALUE

 ROW#2            0                  5/8               1/2

 ROW#3            0                 -1/8               7/2

? END
    .223 CP SECONDS EXECUTION TIME
```

4.3 Session 2: An Advanced Example of EZLP

```
 EZLP - VERSION 9/17/76
 TYPE HELP IF YOU HAVE QUESTIONS, OTHERWISE PROCEED
? MIN: 4.3 A - 5.6 B ++ 7 C
? ST: A + C = B + 7
? AND BOUNDA: A <+ C + 7
? CHANGE BOUNDA "<+"<="
    AND BOUNDA: A <= C + 7
? AND BOUNDB: 1 <= B <= 6
? AND: A>= 0
? USE UPPER PRIMAL


MIN: 4.3 A - 5.6 B ++ 7 C
                    A

                    1
     FATAL ERROR #  1 : ARITHMETIC OPERATOR IN WRONG PLACE

                 **  UNRESTRICTED  VARIABLES  **

  C


     FATAL ERRORS IN THE MODEL.
     PLEASE EDIT THE MODEL
     INPUT WILL BE RECOMPILED
? LIST
 ROW#1
     MIN: 4.3 A - 5.6 B ++ 7 C
 ROW#2
     ST: A + C = B + 7
 BOUNDA
     AND BOUNDA: A <= C + 7
 BOUNDB
     AND BOUNDB: 1 <= B <= 6
 ROW#5
     AND: A>= 0
? CHANGE ROW#1 "++"+"
     MIN: 4.3 A - 5.6 B + 7 C
? PRINT FINAL ALL
? USE UPPER PRIMAL

                 **  UNRESTRICTED  VARIABLES  **

  C
              **  VARIABLE LIST  **


  A          B          C
```

```
PHASE 2 ITERATION -     1  CURRENT OBJ VALUE = .3015E+02
   PRIMAL NON-ZERO VARIABLES
    A          BASIC WITH VALUE  .7500E+01
    B          NON-BASIC WITH VALUE  .1000E+01
    C          BASIC WITH VALUE  .5000E+00
   DUAL NON-ZERO VARIABLES
    ROW#2      DUAL =  .565E+01
    BOUNDA     DUAL = -.135E+01
THE CURRENT BASIC VARIABLES ARE -
    C          A
CURRENT TABLEAU -
            RHS        A           B          C          SLK#3
ROW#1      -.302E+02 0.          .500E-01 0.           .135E+01
C           .500E+00 0.         -.500E+00  .100E+01 -.500E+00
A           .750E+01  .100E+01 -.500E+00 0.           .500E+00
UPDATED OBJECTIVE FUNCTION ROW
                     0.         -.500E-01 0.          -.135E+01



         S O L U T I O N
         ---------------

  OBJECT            MINIMIZE  ROW#1
  Z                       .3015E+02
  ITERATIONS            1

 V A R I A B L E   S E C T I O N
 -------------------------------
NAME        ACTIVITY LEVEL    LOWER BOUND     UPPER BOUND    REDUCED COST

A              .7500E+01      0.               .1000E+22     0.

B              .1000E+01       .1000E+01       .6000E+01      .5000E-01

C              .5000E+00      -.1000E+22       .1000E+22     0.


  C O N S T R A I N T   S E C T I O N
  -----------------------------------
NAME      SLACK ACTIVITY          DUAL PRICE        RHS VALUE

ROW#2         0.                .5650E+01       .5000E+00

BOUNDA        0.               -.1350E+01       .7500E+01

? END
     .400 CP SECONDS EXECUTION TIME
```

4.4 Session 3: A Transportation Problem Example

```
 EZLP - VERSION 9/17/76
 TYPE HELP IF YOU HAVE QUESTIONS, OTHERWISE PROCEED
? TITLE  TRANSPORTATION PROBLEM - SOLUTION VIA EZLP.
? MAX TRANSOBJ: 2X1,1 + 3X1,2 + 4X2,1 + 2X2,2
? ST  SUPPLY1:   X1,1 +  X1,2                    = 3
? AND SUPPLY2:                    X2,1 +  X2,2 = 4
? AND DEMAND1:   X1,1          +  X2,1           = 5
? AND DEMAND2:           X1,2           +  X2,2 = 2
? AND: ALL VARS >= 0
? PRINT BASIS
? USE RATIONAL PRIMAL

      ** NO UNRESTRICTED VARIABLES IN THE MODEL **

               **  VARIABLE LIST  **


  X1,1      X1,2      X2,1      X2,2

 PHASE 1 ITERATION -     1  CURRENT OBJ VALUE =    14
 THE CURRENT BASIC VARIABLES ARE -
     ART#2     ART#3     ART#4     ART#5

 PHASE 1 ITERATION -     2  CURRENT OBJ VALUE =    8
 AT THIS ITERATION, X1,1      ENTERED THE BASIS AND ART#2    LEFT
 THE CURRENT BASIC VARIABLES ARE -
     X1,1      ART#3     ART#4     ART#5

 PHASE 1 ITERATION -     3  CURRENT OBJ VALUE =    4
 AT THIS ITERATION, X2,1      ENTERED THE BASIS AND ART#4    LEFT
 THE CURRENT BASIC VARIABLES ARE -
     X1,1      ART#3     X2,1      ART#5

 PHASE 1 ITERATION -     4  CURRENT OBJ VALUE =    0
 AT THIS ITERATION, X1,2      ENTERED THE BASIS AND ART#3    LEFT
 THE CURRENT BASIC VARIABLES ARE -
     X1,1      X1,2      X2,1      ART#5

 PHASE 2 ITERATION -     1  CURRENT OBJ VALUE =    24
 THE CURRENT BASIC VARIABLES ARE -
     X1,1      X1,2      X2,1      ART#5
```

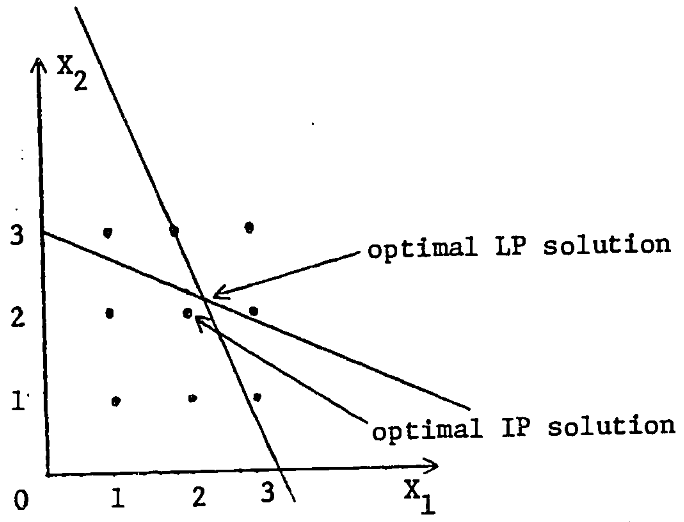
```
TRANSPORTATION PROBLEM - SOLUTION VIA EZLP.


       S O L U T I O N
       ---------------

OBJECT              MAXIMIZE  TRANSOBJ
Z                            24
ITERATIONS             1

 V A R I A B L E   S E C T I O N
 -------------------------------
NAME      ACTIVITY LEVEL    LOWER BOUND    UPPER BOUND   REDUCED COST

X1,1            1                0            POS INF         0

X1,2            2                0            POS INF         0

X2,1            4                0            POS INF         0


  C O N S T R A I N T   S E C T I O N
  -----------------------------------
NAME      SLACK ACTIVITY          DUAL PRICE       RHS VALUE

SUPPLY1         0                    3                1

SUPPLY2         0                    5                2

DEMAND1         0                   -1                4

DEMAND2         0                    0                0

? END
       .413 CP SECONDS EXECUTION TIME
```

4.5 Session 4: A Branch and Bound Example

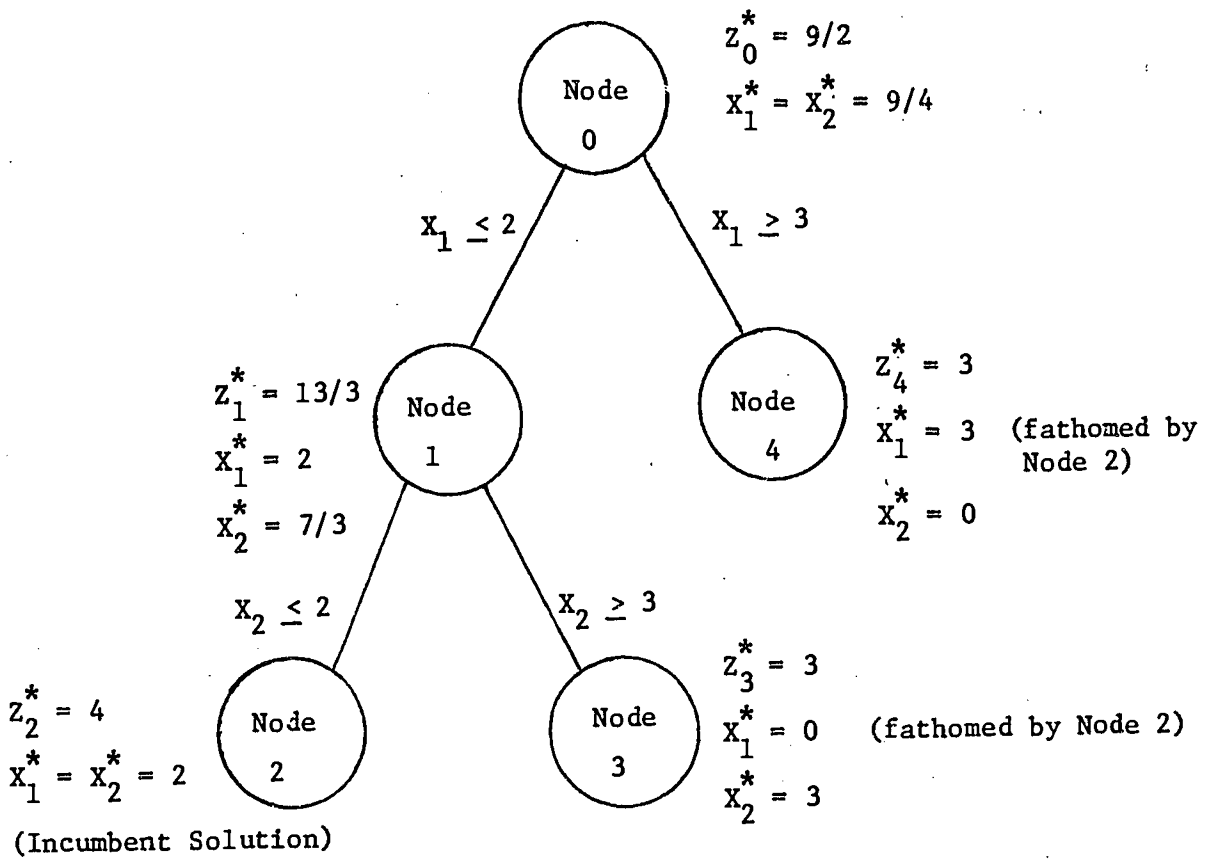
If we consider the integer program

$$\begin{aligned} \text{Max} \quad & X_1 + X_2 \\ \text{ST} \quad & X_1 + 3X_2 \leq 9 \\ & 3X_1 + X_2 \leq 9 \\ & X_1, X_2 \geq 0 \text{ and integer} \end{aligned}$$

described by the graph



Using EZLP, we can construct the following Branch and Bound tree:



4.6 Session 5: A Sensitivity Analysis Example

```
     EZLP - VERSION 9/17/76
     TYPE HELP IF YOU HAVE QUESTIONS, OTHERWISE PROCEED
    ? TITLE  NODE 0 - SOLUTION VIA EZLP
    ? MAX: X1+X2
    ? ST: X1+3X2 <= 9
    ? AND: 3X1+X2 <= 9
    ? AND: ALL VARS >= 0
    ? USE RATIONAL UPPER PRIMAL

          ** NO UNRESTRICTED VARIABLES IN THE MODEL **

                  **  VARIABLE LIST  **


     X1        X2


       NODE 0 - SOLUTION VIA EZLP


             S O L U T I O N
             ---------------

       OBJECT           MAXIMIZE  ROW#1
       Z                         9/2
       ITERATIONS           3

      V A R I A B L E   S E C T I O N
      -------------------------------
     NAME      ACTIVITY LEVEL   LOWER BOUND   UPPER BOUND   REDUCED COST

     X1              9/4            0            POS INF          0

     X2              9/4            0            POS INF          0


       C O N S T R A I N T   S E C T I O N
       -----------------------------------
     NAME      SLACK ACTIVITY          DUAL PRICE      RHS VALUE

     ROW#2          0                  1/4               9/4

     ROW#3          0                  1/4               9/4

    ? AND: X1<=2
    ? TITLE  NODE 1 - ADDITION OF ONE BOUND ON X1
    ? USE RATIONAL UPPER PRIMAL

          ** NO UNRESTRICTED VARIABLES IN THE MODEL **

                  **  VARIABLE LIST  **


     X1        X2
```

```
  NODE 1 - ADDITION OF ONE BOUND ON X1


          S O L U T I O N
          ---------------

  OBJECT              MAXIMIZE  ROW#1
  Z                               13/3
  ITERATIONS             3

 V A R I A B L E   S E C T I O N
 -------------------------------
NAME       ACTIVITY LEVEL    LOWER BOUND     UPPER BOUND    REDUCED COST

X1                2              0                2             2/3

X2               7/3             0             POS INF          0

SLK#3            2/3             0             POS INF          0


  C O N S T R A I N T   S E C T I O N
  -----------------------------------
NAME       SLACK ACTIVITY               DUAL PRICE       RHS VALUE

ROW#2         0                            1/3              7/3

ROW#3        2/3                           0                2/3

? AND: X2 <= 2
? TITLE  NODE 2 - UPPER BOUNDS ON X1, X2
? USE RATIONAL UPPER PRIMAL

      ** NO UNRESTRICTED VARIABLES IN THE MODEL **

               **  VARIABLE LIST  **


 X1         X2


  NODE 2 - UPPER BOUNDS ON X1, X2


          S O L U T I O N
          ---------------

  OBJECT              MAXIMIZE  ROW#1
  Z                                4
  ITERATIONS             3

 V A R I A B L E   S E C T I O N
 -------------------------------
NAME       ACTIVITY LEVEL    LOWER BOUND     UPPER BOUND    REDUCED COST

X1                2              0                2             1

X2                2              0                2             1

SLK#2             1              0             POS INF          0

SLK#3             1              0             POS INF          0
```

```
 C O N S T R A I N T   S E C T I O N
 -----------------------------------
NAME       SLACK ACTIVITY          DUAL PRICE      RHS VALUE

ROW#2            1                  0                1

ROW#3            1                  0                1

? LIST
 ROW#1
    MAX: X1+X2
 ROW#2
    ST: X1+3X2 <= 9
 ROW#3
    AND: 3X1+X2 <= 9
 ROW#4
    AND: ALL VARS >= 0
 ROW#5
    AND: X1<=2
 ROW#6
    AND: X2 <= 2
? CHANGE ROW#6 '<= 2'>= 3'
    AND: X2 >= 3
? TITLE  NODE 3 - LOWER BOUND ON X2,  UPPER BOUND ON X1
? USE RATIONAL UPPER PRIMAL

      ** NO UNRESTRICTED VARIABLES IN THE MODEL **

               **  VARIABLE LIST  **


 X1          X2


  NODE 3 - LOWER BOUND ON X2,  UPPER BOUND ON X1


          S O L U T I O N
          ---------------

  OBJECT              MAXIMIZE  ROW#1
  Z                              3
  ITERATIONS             2

 V A R I A B L E   S E C T I O N
 -------------------------------
NAME       ACTIVITY LEVEL    LOWER BOUND    UPPER BOUND    REDUCED COST

X2               3               3            POS INF          2

SLK#3            6               0            POS INF          0


 C O N S T R A I N T   S E C T I O N
 -----------------------------------
NAME       SLACK ACTIVITY          DUAL PRICE      RHS VALUE

ROW#2            0                  1                0

ROW#3            6                  0                6
```

```
? DELETE ROW#6
? LIST ROW#5
 ROW#5
     AND: X1<=2
? CHANGE ROW#5 "<=2">=3"
     AND: X1>=3
? TITLE  NODE 4 - LOWER BOUND ON X1 (FINAL NODE)
? USE RATIONAL UPPER PRIMAL

      ** NO UNRESTRICTED VARIABLES IN THE MODEL **

                 **  VARIABLE LIST  **


  X1          X2


   NODE 4 - LOWER BOUND ON X1 (FINAL NODE)


          S O L U T I O N
          ---------------

   OBJECT             MAXIMIZE  ROW#1
   Z                           3
   ITERATIONS              3

  V A R I A B L E  S E C T I O N
  ------------------------------
  NAME        ACTIVITY LEVEL     LOWER BOUND     UPPER BOUND    REDUCED COST

  X1                   3             3              POS INF          2

  SLK#2                6             0              POS INF          0


   C O N S T R A I N T  S E C T I O N
   ----------------------------------
  NAME        SLACK ACTIVITY              DUAL PRICE        RHS VALUE

  ROW#2              6                       0                 6

  ROW#3              0                       1                 0

 ? LIST
  ROW#1
      MAX: X1+X2
  ROW#2
      ST: X1+3X2 <= 9
  ROW#3
      AND: 3X1+X2 <= 9
  ROW#4
      AND: ALL VARS >= 0
  ROW#5
      AND: X1>=3
 ? END
      1.152 CP SECONDS EXECUTION TIME
 EZLP - VERSION 12/6/79 - RANGE TEST
 TYPE HELP IF YOU HAVE QUESTIONS, OTHERWISE PROCEED
? MAX: 3X1+4X2
? ST: X1<=3
? AND: X2<=5
? AND: X1+X2<=7
? AND: X1,X2>=0
? USE UPPER PRIMAL
 VARIABLE LIST -
 X1        X2
 NO UNRESTRICTED VARIABLES IN THE MODEL


          S O L U T I O N
          ---------------

   OBJECT             MAXIMIZE  ROW#1
   Z                          .2600E+02
   ITERATIONS           3

  V A R I A B L E   S E C T I O N
  -----------------------------
 NAME       ACTIVITY LEVEL    LOWER BOUND    UPPER BOUND    REDUCED COST

 X1             .2000E+01     0.              .3000E+01      0.

 X2             .5000E+01     0.              .5000E+01       .1000E+01


   C O N S T R A I N T   S E C T I O N
   ---------------------------------
 NAME       SLACK ACTIVITY     DUAL PRICE      RHS VALUE

 ROW#4          0.              .3000E+01      .2000E+01

? RANGE ALL

RANGE INFORMATION

 CONSTRAINT NAME     LOW        CURRENT      HIGH

      ROW#4      .50000E+01  .70000E+01  .80000E+01

 VAR TYPE       NAME          LOW        CURRENT      HIGH

BASIC    ROW#4   X1       0.              .30000E+01  .40000E+01
NON-BASIC        X2         .30000E+01  .40000E+01  .10000E+21
NON-BASIC        SLK#2    -.10000E+21 0.              .30000E+01
? END
      .320 CP SECONDS EXECUTION TIME
```

APPENDIX A

A SIMPLIFIED USER'S MANUAL FOR EZLP

The first three pages of this appendix serve as a reasonable handout for most beginning users. The remaining nine pages provide more detailed discussion of the topics covered in the first three pages. By employing the "HELP" command the user can obtain any or all of this information during execution of EZLP.

EZLP: AN INTERACTIVE COMPUTER PROGRAM DESIGNED TO SOLVE STUDENT-ORIENTED LINEAR PROGRAMMING PROBLEMS.†

1: General

The first word of each line is a keyword, and indicates the function of the input line. The keywords are "MIN" or "MAX" for the objective function, "ST" for the first constraint, followed by "AND" for each additional constraint. Each name (e.g. variable name) consists generally of a combination of up to 8 alphabetic and numeric characters with the first character being alphabetic.

Model Entry

Model entry statements are composed of a keyword, an optional name, a mandatory colon, and either an objective function or a constraint.

Example of a model entry:

```
MAX:  2PROD1 + 3PROD2 - 4 COST
ST:   PROD1 + PROD2<=150
AND:  COST >= 5
AND:  ALL OTHER VARS >= 0
```

Editing

The input model can be edited by "DELETE" and "CHANGE" statements. Adding a constraint is accomplished by simply typing the constraint.

Examples:

```
AND NEWROW:  5COST-PROD1>=20.23
CHANGE NEWROW "PROD1"PROD2"
DELETE NEWROW
```

†EZLP was developed in the School of Industrial and Systems Engineering, Georgia Tech, Atlanta, Georgia 30332 under NSF grant #SED75-17476. John J. Jarvis, project director; Frank H. Cullen and Chris Papaconstadopoulos, research assistants.

EZLP internally numbers the input model lines and if the optional name before the colon is omitted during model entry, a name of the form "ROW#n" (e.g., ROW#6) is assigned as a default.

The model can be listed either in total or one line at a time as follows:

```
LIST                    (THE ENTIRE MODEL)
LIST CONSTR2          (LINE"CONSTR2" ONLY)
```

Continuation of the input line is achieved by placing an ampersand (&) after a complete name. Example:

```
AND:   HEAT - 2.34WOOD &
     -6.45POWER>=16.4
```

Specifying Output

EZLP always outputs the final optimal solution. Other output can be requested after every iteration or after every n iteration. Examples:

```
PRINT BASIS 5 (The names of the basic variables every
               fifth iteration)
PRINT TABLEAU, 2 (The tableau at every other iteration)
PRINT VARS    (The value of primal and dual variables)
PRINT NONE    (Resets print specs to only final solution)
PRINT INITIAL, FINAL TABLEAU
PRINT ALL     (Prints everything)
```

Specifying the Method of Solution

After model entry, editing, and output specification, the USE statement triggers the optimization. Examples:

```
USE PRIMAL
     (Ordinary simplex method)
USE RATIONAL PRIMAL
     (Ordinary simplex method with rational arithmetic)
USE UPPER DUAL
     (Lower-upper bounded dual simplex method)
USE RATIONAL UPPER PRIMAL
     (Lower-upper bounded simplex method with rational
      arithmetic)
```

Upon completion of the solution attempt, EZLP returns to the editing phase.

Sensitivity Analysis

After executing any of the USE commands to obtain an optimal solution, the user may obtain sensitivity information by use of the RANGE command. Application of the range command determines lower and upper limits on the objective coefficients and/or right-hand-side constants which keep the current optimal basis.

Examples:

```
RANGE     RHS     ROW#6
RANGE     OBJ     X54
RANGE     OBJ     ALL
RANGE     ALL
```

Restarting and Stopping EZLP

If another model is to be entered from scratch, type "RESTART", and EZLP will re-initialize the model file area. If EZLP is to be terminated, type "END".

File-Handling Capabilities

EZLP has the ability to use external mass storage files for both input and output. Information on these options can be obtained by consulting Section 5 "FILE-HANDLING".

Simple Example of a Complete EZLP Run

```
MAX:  2X1+3X2+4.5X3
ST:   X1+X2              <=75.3
AND:     X2  +X3         <=45
AND:  ALL VARS >=0
USE PRIMAL
END
```

The Help Command

During execution EZLP permits the user to obtain specific information concerning its use. The user may obtain a short introduction to EZLP and its use by typing

```
HELP
```

If the user desires more specific instructions, he may type "HELP<keyword>", (e.g., HELP MODEL) where the acceptable keywords are:

General keywords

EDIT	EXAMPLES	FILES	KEYWORDS
MODEL	SOLVING	BRIEF	

Specific keywords

ADD	ADVANCED	CHANGE	CONT
CONS	DELETE	OBJ	PRINT
RUN	SAVE	TITLE	USE
ALTOBJ	NAMES	LIST	

2: The Model

Providing a Title for the Model

EZLP permits the input of a title for the model. This title will appear just before the optimal solution is printed. The title command consists of:

1. The keyword "TITLE"
2. The title text

Example:

```
TITLE THIS IS MODEL 4
```

Variable and Constraint Names

In entering the model, the user has the ability of assigning his own names to the variables and the constraints. These names must be from one to eight characters long, begin with an alpha character, and contain no special characters. A variable name may contain a comma provided that the comma separates two numerics. Spaces are not allowed within names. EZLP generates constraint names for unnamed rows and variable names for slack and artificial variables. These take the form:

ROW#n - for generated constraint names
SLK#n - for generated slack variable names
ART#n - for generated artificial variable names

Continuation of Lines

In the event that an input line exceeds 80 characters, continuation is accomplished by placing an ampersand (&) after a complete name and resuming input on the following line.

Objective Functions

EZLP allows for the specification of two kinds of objective functions: primary and alternate. For the simple model there usually will not be an alternate objective function. In essence, this concept allows the user to enter more than one objective function and optimize the various objective functions subject to the same constraints during the course of the interactive session.

Primary objective function - The entry of the objective function consists of:

1. The keywords "MAX", "MIN"
2. An optional objective function name
3. A mandatory colon
4. And a linear combination of user-defined variable names.

If the optional objective function is omitted, EZLP will generate a name of the form ROW#n for use in editing and output. Examples:

```
MIN:  2X1 + 2X2 + 3X3
MAX MYOBJ:  3.54 COLUMN1-4.543COLUMN2 + 6.25
MINIMIZE:  5.76 X1,2 - 4.33 X1,3 + 5.45 X2,1 & - 6.43 X2,3
```

Alternate Objective Functions - Alternate objective functions are entered in the same way as the primary objective function, except that the keyword "MIN" or "MAX" is replaced by the keyword "ALSO". The use of an alternate objective function is discussed in Section 4 "Solving the Model". Examples:

```
ALSO:  2X1 - 3X2 - 4X3
ALSO OBJ2:  3.45 COLUMN1 - 4.511COLUMN2
```

Constraints

Constraints are divided into three generic types.

1. Simple arithmetic constraints
2. Range constraints
3. List constraints

Each is discussed below.

Simple Arithmetic Constraints

A simple arithmetic constraint consists of

1. The keyword "AND" or "ST"
2. An optional constraint name
3. A mandatory colon
4. A linear combination of user-defined variable names
5. A relational operator (=,<=,>=)
6. A second linear combination of user-defined variable names.

It is not allowable to have the same variable name in both 4 and 6 above.

Examples:

```
AND:  X1+X2 =7
AND MYCSTR:  5X1 + 8.3 X2 >= X3 +4
```

Range Constraints

EZLP allows the entry of bounded arithmetic expressions (range constraints). Range constraints consist of the following:

1. The keyword "AND" or "ST"
2. An optional user-defined constraint name
3. A mandatory colon
4. A constant (constant-1)
5. An inequality relational operator (>= or <=)
6. A linear combination of the user-assigned
7. An inequality operator (>= or <=)
8. A constant (constant-2)

The restriction on range constraints are that the two inequality operators must be identical and that the two constants must be consistent. (i.e. if the inequality operators are <=, constant-1 must be <= constant-2).

```
AND:  4 <= 3X1+4.3X2<= 6.2

AND BOUND1:  7.2>= 3X2+HEAT - 5POWER>= 5
```

List Constraints

EZLP allows bounds for collective groups (or lists) of user-defined variable names to be specified in list constraints. The variable lists can be either explicit or implicit.

Explicit List Constraints

(These constraints consist of:

1. The keyword "AND"
2. An optional constraint name
3. A mandatory colon
4. A list of variable names
5. A relational operator
6. A constant

The variable list is simply a list of variable names separated by commas. If the "UPPER" option is not specified in the "USE" statement, those list constraints in which the constant is not 0 are considered as explicit rows in the simplex tableau. Examples:

```
AND:  X1,X2,X5 >=0
AND UBND:  HEAT, POWER, LIGHT <=100.3
AND:  MYVAR, YOURVAR URS
      (Here URS = unrestricted in sign and takes the
       place of 5 and 6 above.)
```

Bounded lists are also permitted, for example

```
AND:  4< = X1,X2,X5< =7
```

<u>Implicit List Constraints</u>

Special abbreviations are available to describe either all variables or variables not appearing in another list constraint. Examples:

```
AND:  ALL VARS> = 0
AND:  ALL VARS< = 1
AND MYBOUND:  ALL OTHER VARS >= 3
```

The phrase "ALL VARS" is a synonym for "all variables", while the phrase "ALL OTHER VARS" refers only to those variables not included in some earlier list constraint.

3: Editing the Model

Adding Rows

New rows may be either appended to the current model file or inserted within the current model file.

a. Appending rows to the end of the model. This is easily accomplished by simply typing in the row exactly as it was done during model entry. Example:

```
AND:  X1+X2 <= 7
ALSO:  3.5X1+4.6FLOW1
```

b. Insertion of rows within the model file - Rows may be inserted within the model file by simply typing in:

```
INSERT AFTER <old-row>  <model entry statement>
```

or

```
INSERT BEFORE <old-row>  <model entry statement>
```

For example, if the current model file contained:

```
MIN OBJ:  2X1+3X2
ST CON:  X1+X2<=3
AND:  ALL VARS >=0
```

an entry of

```
INSERT AFTER CON AND CON2:  X1+2X2 >=1
```

would result in a model file of

```
MIN OBJ:  2X1+3X2
ST CON:  X1+X2<=3
AND CON2:  X1+2X2 >=1
AND:  ALL VARS >=0
```

Deleting Rows

The deletion of a row is accomplished by entering DELETE <row name> where <row name> is the user-assigned name, or in the absence of such, takes the form "ROW#n". If a delete is followed by the addition of a row having the same name, the new row is inserted in the model file in the same place. Examples:

```
DELETE MYCSTRNT
DELETE ROW#7
```

an entry of

```
CHANGE ROW#5 "3"X2+4"
```

would result in ROW#5 becoming

```
AND:  2X1+X2+4X3<=5
```

whereupon an entry of

```
CHANGE ROW#5 "3<""
```

would yield ROW#5 as

```
AND:  2X1+X2+4X=5
```

Changes involving row names and/or continuation characters (&) are illegal. Changes of this type should be accomplished by a "DELETE" followed by retyping the entire row.

<u>Listing the Model</u>

The list command allows the user to list the current contents of the model file in total or in part. Two options are available:

1. LIST	(lists the whole model)
2. LIST <row name>	(lists only row <row name>)

Examples:

```
LIST
LIST ROW#5
LIST CONSTR3
```

<u>Changing Rows</u>

To change a character string in a specified row (say ROW#5), the following is entered

```
CHANGE ROW#5 "<string1>"<string2>"
```

The double quote (") is the only allowable delimiter. For example if ROW#5 was originally

```
AND:  2X1+3X3<=5
```

4: Solving the Model

Specifying Alternate Objective Functions

The user may specify an alternate objective function to be optimized by typing:

```
ALTOBJ MIN  <row name>
```

or

```
ALTOBJ MAX  <row name>
```

where <row name> is the name of an objective function in the model.

Examples:

```
ALTOBJ MAX ROW#2
ALTOBJ MIN OBJ4
```

This objective function specification remains in force until such time as a "RESTART" or another "ALTOBJ" command is entered.

The user may identify the current specified alternate objective function row name by typing:

```
ALTOBJ STATUS
```

Specifying Output

The PRINT command is used to specify the type and frequency of output to be printed during the optimization process. Parameters which can be used are:

1.	VARS	prints the values of the non-zero primal and dual variables in the current tableau.
2.	BASIS	prints the current basic variables and denotes entry and exit activities
3.	TABLEAU	prints the current tableau and the updated objective function row
4.	ALL	Synonym for "VARS,BASIS,TABLEAU"

Parameters 1-4 can optionally be followed by an integer indicating the frequency with which the output is to be given. If omitted, the default value is 1 (output after every iteration). Examples:

```
PRINT TABLEAU, BASIS
PRINT VARS 5
PRINT ALL 4
```

Initial and/or final tableau output can also be indicated. In this case, the integer described does not apply and is ignored if present. Examples:

```
PRINT INITIAL TABLEAU
PRINT FINAL ALL
PRINT INITIAL, FINAL BASIS TABLEAU
```

If print specifications are to be altered, the proper action is to enter

```
PRINT NONE
```

which resets print parameters to the default mode.

Specifying the Method of Solution

The USE command is entered when an attempt to solve the input model is to be made. The following parameters are legal (the order of the parameters is important).

1. RATIONAL - (Optional) Rational arithmetic is to be used. Rational arithmetic is 3 to 4 times slower than real arithmetic, and can only solve problems half the size.

2. UPPER - (Optional) Lower-upper bounded simplex is to be used. List constraints are implicit in the tableau.

3. PRIMAL or DUAL - A specification of the type simplex method to be used. "PRIMAL" refers to the simplex method and "DUAL" refers to the dual simplex method.

Examples:

```
USE RATIONAL PRIMAL
USE DUAL
USE UPPER DUAL
USE RATIONAL UPPER PRIMAL
```

When possible, "UPPER" should be specified for dual simplex, as this improves the chance of obtaining a starting dual feasible basis.

5: Sensitivity Analysis

EZLP contains capability for sensitivity analysis of an optimal solution of a model. This is accomplished by use of a RANGE command. The format of this command is:

1. The keyword "RANGE".
2. User defined options for ranging: "RHS", "OBJ" or "OBJFCN", "ALL".
3. Optional user defined variable names or row names, or the optional modifier "ALL".

EXAMPLES:

```
RANGE    RHS    ROW#4
RANGE    OBJ
RANGE    OBJ    X55
RANGE    OBJ    ALL
RANGE    ALL
```

6: File-Handling

Input From an External File

The "RUN" command enables the user to use an external file as a store of input statements and commands. Upon entering the command RUN <file name>, EZLP opens the local file <file name> and references this file for all subsequent input commands, until such time as an "END" statement is encountered or an end-of-file condition on <file name> exists. If an end-of-file is encountered, control returns to the live user. The form of the file should be 80 characters per line.

Output to an External File

The "SAVE" command enables the user to place the current concepts of the model file into a local file. The form of this file is 80 characters per line and editable by the system editor or usable as input to a later execution of EZLP. ENTER:

```
SAVE <file name>
```

6: Batch Processing

EZLP can be used in a batch environment in much the same way that it is used interactively. The sole restriction is that the command BAT must be the first entry in the input deck to EZLP. Omission of this command can result in an infinite print loop at job termination time. Aslo the command END must be the last entry in the input deck to EZLP.

7: Advanced Example

```
TITLE ** ADVANCED MODEL **
MAX OBJ:  2INTERST1+3.24INTERST2-5CAPITAL
ALSO OBJ2:  3.4INTERST1+2.54INTERST2-5 CAPITAL
ST CONSTR1:  INTERST1-4INTERST2 = 108
AND MINCAP:  CAPITAL> = 4.6 INTERST1 + 5.7INTERST2
AND MAXCAP:  4 CAPITAL< = 10345.65 - 3.224 INTERST1 &
             -5.231 INTERST2
AND CAPBND:  800  <=CAPITAL  <= 10000
AND INTBND:  INTERST1, INTERST2 <=10825
AND NONNEG:  INTERST1, INTERST2 >= 0
CHANGE MINCAP "4.6"4.76"
CHANGE INTBND "825".825"
PRINT ALL
USE UPPER PRIMAL
DELETE NONNEG
AND NEWCSTR:  INTERST2  >= 1.342
AND UBND:  INTERST1, INTERST2,CAPITAL  <= 100000.3
ALTOBJ MAX OBJ2
PRINT FINAL TABLEAU, BASIS, VARS
LIST
USE UPPER DUAL
END
```