LPL : A PROBLEM ORIENTED LANGUAGE FOR LINEAR PROGRAMMING

By

EVANGEL P. QUIWA*

Introduction

Linear programming is a mathematical technique used to determine the optimal allocation of limited resources, such as men, money, materials and machines to satisfy a given objective, e.g., minimize cost or maximize profit. In the last decade since the general linear programming (LP) problem was formulated and the simplex method developed for its solution, linear programming has become an indispensable tool in solving certain optimization problems in such diverse fields as economics, military operations and engineering.

The simplex algorithm for solving the LP problem is essentially a simple method. One who knows how to add, subtract, multiply and divide can readily apply the procedure. Of course, to understand why the procedure works requires more than just a knowledge of arithmetic. In any case, the large amount of repetitive calculations needed to solve a typical LP problem makes it impractical (and for really large problems, impossible) to carry out the computations by hand. A digital computer, not a human person, is better suited to perform this kind of tedious chore. However, a computer must first be instructed in the ways of linear programming.

Described herein is a program package called LPL, acronym for linear Programming Language, which is designed specifically to solve the LP problem. LPL is currently implemented on the IBM/360-256K system of the U.P. Computer Center and is available for general use. Some of the salient features of LPL are:

1. LPL accepts the LP problem in "raw" form, i.e., as formulated. Slack, surplus and artificial variables are automatically provided as needed.

*Associate Professor, Engineering Sciences Department, University of the Philippines.

2. Input of commands and numerical data is free-field. The user, however, may input the activity matrix (which constitute the bulk of the numerical data) in either free-field or fixed-field fashion.

3. The user may choose between two implementations of the simplex algorithm: the big-M method and the two-phase method.

4. The user can obtain an iteration by iteration printout of all computational results in tableau form, or an iteration log only, or both.

5. The user can readily modify the original input data and solve the problem anew.

6. The user can solve the dual problem, subject only to the restriction that there are no equation-type primal constraints.

7. The user can perform a sensitivity analysis on the objective function coefficients and the right-hand-side values.

These features of LPL make it highly suitable for instructional use.

In the next two sections of this paper, LPL will be described from two points of view: firstly, LPL as a language — its alphabet, statements, program structure, error messages and the like; and, secondly, LPL as an interpretive system — its provisions for command interpretation and the free-field input of both numeric and non-numeric data.

LPL : Its Features as a Language

The format of LPL statements and the structure of LPL programs are closely related to the form of the LP problem. It is therefore instructive to review at this point the general linear programming problem and some LP terminology. Mathematically, the LP problem may be defined as follows:

Optimize the objective function

$$Z = \sum_{j=1}^{n} c_j x_j \qquad (1)$$

Subject to the constraints

$$\sum_{j=1}^{n} a_{ij} x_j \begin{matrix} > \\ = \\ < \end{matrix} b \qquad i = 1,2,\ldots,m \qquad (2)$$

and the non-negativity restrictions

$$x_j > 0 \qquad j = 1,2,\ldots,n \qquad (3)$$

The quantities $x_j, j = 1,2,\ldots,n$, are called structural variables; the coefficients c_j of the x_j in the objective function will be referred to as

"prices". The quantities $a_{ij}, j = 1,2,\ldots,n;\ i = 1,2,\ldots,m$, are called activity or technological coefficients, while the b_i, $i = 1,2,\ldots,m$, are called resource coefficients or, simply, RHS values. The values of all the c_j, a_{ij} and b_i are assumed to be known.

Any set of x_j which satisfies the constraints (2) is a solution to the LP problem. Any solution which satisfies the non-negativity restrictions (3) is called a feasible solution. Any feasible solution which optimizes the objective function (1) is called an optimal feasible solution.

Consider now a specific linear programming problem (a student project in ES 240, now CE 218, Operations Research in Water Resources):

$$\begin{aligned}\text{Maximize } V = {} & 51.48\,x_{11} + 25.92\,x_{12} + 46.66\,x_{21} + 21.60\,x_{22} \\ & + 41.47\,x_{31} + 19.44\,x_{32} + 60.48\,x_{41} + 30.24\,x_{42} \\ & + 65.66\,x_{51} + 36.72\,x_{52} + 69.12\,x_{61} + 43.20\,x_{62}\end{aligned}$$

Subject to

$$\begin{array}{rl}
\text{AQSL:} & x_{11} + x_{12} \leqslant 2100 \\
\text{ASFCL:} & x_{21} + x_{22} \leqslant 405 \\
\text{ASFC:} & x_{31} + x_{32} \leqslant 80 \\
\text{ALPS:} & x_{41} + x_{42} \leqslant 715 \\
\text{ALDFS:} & x_{51} + x_{52} \leqslant 500 \\
\text{AACS:} & x_{61} + x_{62} \leqslant 200 \\
\text{MAC:} & x_{12} + x_{22} + x_{32} + x_{42} + x_{52} \\
 & + x_{62} > 500 \\
\text{AWC:} & 30x_{11} + 12x_{12} + 27x_{21} + 10x_{22} + 24x_{31} \\
 & + 9x_{32} + 35x_{41} + 14x_{42} + 38x_{51} \\
 & + 17x_{52} + 40x_{61} + 20x_{62} < 120000 \\
\text{MIPC:} & 9(x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + \\
 & x_{61}) \\
 & +6(x_{12} + x_{22} + x_{32} + x_{42} + x_{52} + \\
 & x_{62} < 40000
\end{array}$$

$$\text{MBC:}\quad 40(x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{61}$$
$$x_{61}$$
$$+22.5\,(x_{12} + x_{22} + x_{32} + x_{42} + x_{52} + x_{62})$$
$$< 129000$$

The following LPL program will solve this (primal) problem and also its dual. In addition, a sensitivity analysis will be performed on the objective function coefficients and RHS values for both the primal and dual problems.

```
TASKNAME FSDC PUMP IRRIGATION PROJECT (STUDENT PROJECT IN
    IN ES 240)
VARIABLES  x11(51.48) x12(25.92) x21(46.66) x22(21.60) x31(41.47)
*x32(19.44) x41(60.48) x42(30.24) x51(65.66) x52(36.72)
*61(69.12) x62(43.20)

FORMAT FREE MATRIX

CONSTRAINTS AQSL ≤ 2100 ASFCL ≤ 405 ASFC ≤ 80 ALPS X 715      X 715
*ALPS ≤ 500  AACS ≤ 200  MAC ≤ 500  AWC ≤ 120000 MIPC
    ≤ 40000
*MBC ≤ 129000

   1   1     0   0     0   0     0   0     0   0     0   0
   0   0     1   1     0   0     0   0     0   0     0   0
   0   0     0   0     1   1     0   0     0   0     0   0
   0   0     0   0     0   0     1   1     0   0     0   0
   0   0     0   0     0   0     0   0     1   1     0   0
   0   0     0   0     0   0     0   0     0   0     1   1
   0   1     0   1     0   1     0   1     0   1     0   1
  30  12    27  10    24   9    35  14    38  17    40  20
   9   6     9   6     9   6     9   6     9   6     9   6
  40 22.5   40 22.5   40 22.5   40 22.5   40 22.5   40 22.5

PRINTFORMAT
ECHOPRINT PRIMAL
TABLEAUX FIRST, LAST*
SUMMARY
RANGES ALL
METHOD TWO-PHASE
MAXIMIZE PRIMAL VOLUME
:
: SOLVE THE DUAL PROBLEM
:
RENAME DUAL PROBLEM: FSDC PUMP IRRIGATION PROJECT
ECHOPRINT DUAL
TABLEAUX LAST
MINIMIZE DUAL VOLUME
TASKEND
STOPRUN
```

With this sample LPL program as a point of departure, we will now look at LPL as a language.

A sequence of LPL statements, starting with the TASKNAME statement and ending with the TASKEND statement is an LPL *task*, and is the equivalent of a given LP problem. The TASKNAME and TASKEND statements constitute the task-definition statements of LPL. Immediately following the TASKNAME statement are the data-definition statements, namely, VARIABLES, FORMAT and CONSTRAINTS statements. Through these three statements, the user specifies the numerical data of the LP problem he desires to solve and also the names he wishes to assign to the variables and constraints of the problem.

The next six statements in the sample program belong to the class of statements called specification statements. These statements allow the user to specify additional information which does not pertain to the LP problem as such, but which is needed nevertheless by the processing programs of LPL. For example, the user specifies through the PRINT-FORMAT statement the format code to be used in printing the input problem data (if ECHOPRINT is coded) and the computational results. Printing of intermediate and final results in tableau form is controlled through the TABLEAUX statement. Which implementation of the simplex algorithm (big-M or two-phase) is to be used in solving the LP problem is specified via the METHOD statement. Finally, the RANGES statement instructs the system to perform a sensitivity analysis on the objective function coefficients and RHS values.

The logically last statement in a task is the MAXIMIZE or MINIMIZE statement. With this statement in the program, the specification of the LP problem to be solved is complete. Actually, more than one MAXIMIZE or MINIMIZE statement may appear in a single LPL task. In this case, each such statement logically terminates a *subtask*. In the sample LPL program shown above, the subtask consists in solving the dual problem.

In some cases, it may be desired to delete some variables and/or constraints from the original data as defined by the data-definition statements, or to modify certain numerical values, and then solve the problem anew. Such modification of the input data is accomplished by means of the data-modification statements DELETE, MODIFY and SCALEFACTOR. Application of the simplex algorithm on the modified data also constitutes a subtask.

A sequence of LPL tasks is called a *run*. A run may consist of any number of tasks. Since a task is the equivalent of an LP problem, this means that any number of LP problems may be solved in one LPL run. A run is terminated by the STOPRUN statement.

These concepts and definitions are illustrated in Fig. 1.

```
      / TASKNAME . . .                                                \
      | Data-definition statements                                     |
      | [More LPL statements]            \ a subtask                   > a task
      | MAXIMIZE (or MINIMIZE) . . .     /                             |
      | [More LPL statements]            \ another subtask             |
a run<  MAXIMIZE (or MINIMIZE) . . .     / (there could be more)      /
      | TASKEND                                                       \
      | TASKNAME . . .                                                 |
      | Data-definition statements                                     |  another task
      | [More LPL statements]                                          >  (there could
      | MAXIMIZE (or MINIMIZE)                                         |  be more)
      | TASKEND                                                        |
      \ STOPRUN                                                       /
```

Fig. 1. LPL program structure and related terms

The foregoing paragraphs described the general structure of an LPL program and its relation to the LP problem. In the remainder of this section, the different LPL statements which comprise an LPL program will be described in greater detail under the categories already mentioned.

LPL statements are constructed using characters from the following character set:

(1) Alphabetic and national characters: A,B, ...,Z, P, #
(2) Numeric characters: 1,2, . . .,9,0
(3) Special characters: + - / * . , = : < > ( ) and the blank character.

In addition, LPL labels and comments may contain such other special characters as " ? — ; ' etc.

LPL statements normally consist of two parts: a *control word* (or keyword) which identifies the statement to both the user and the LPL interpreter; and a *specification list* which contains the information to be processed by the appropriate processing routine. This format is illustrated in Fig. 2 (the outline of an 80-column card where LPL statements are punched is also shown).

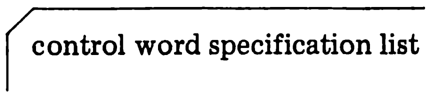

Fig. 2. General format of LPL statements

A. *The task-definition statements*

1. The TASKNAME statement

   a. The general form of the TASKNAME statement is

   ```
   TASKNAME label
   ```

   where *label* is any string of characters. *label* is printed on every page of the output.

   b. The primary function of the TASKNAME statement is to indicate the start of a task. Additionally, it allows the user to specify through *label* the heading that he wants to be printed on every output page.

2. The TASKEND statement

   a. The general form of the TASKEND statement is

   ```
   TASKEND
   ```

   b. The TASKEND statement indicates the (physical) end of a task.

B. *The data-definition statements*

1. The VARIABLES statement

   a. The general form of the VARIABLES statement is

   ```
   VARIABLES    vname (xxx), . . .
   ```

   where *vname* is an LPL name, i.e., a sequence of at most six alphameric characters, the first alphabetic xxx is a constant.

   (and) are required delimiters
   . . . indicates repetition of the same form

   b. The VARIABLES statement identifies to the system (i.e., the processing programs of LPL) the names of the structural variables in the LP problem and their coefficients in the objective function equation (Eq. 1).

c. The specification list of the VARIABLES statement may be continued on as many cards as are necessary, in which case every continuation card must have an * in column 1.

d. The VARIABLES statement must immediately follow the TASKNAME statement. It may not appear more than once in a task.

2. The FORMAT statement

a. The general form of the FORMAT statement is

$$\text{FORMAT} \qquad \begin{Bmatrix}\text{FREE}\\ \text{FIXED}\end{Bmatrix} \qquad \begin{Bmatrix}\text{MATRIX}\\ \text{LIST}\end{Bmatrix}$$

where the braces indicate choice.

b. The FORMAT statement indicates to the system the manner in which the activity matrix (a_{ij}) will be inputted following the CONSTRAINTS statement.

c. If FREE is specified, then input of the data is freefield. If FIXED is specified, then data input will be under the control of a user-specified FORTRAN FORMAT statement.

d. If MATRIX is specified, then the full activity matrix, including zero values, must be inputted rowwise. If LIST is specified, then only the non-zero values need be inputted; however, each such value must be qualified by the appropriate constant and variable names.

3. The CONSTRAINTS statement

a. The general form of the CONSTRAINTS statement is

CONSTRAINTS cname sign xxx

where *cname* is an LPL name

sign is <, = or >

xxx is a constant

. . . indicate repetition of the same form

b. The CONSTRAINTS statement indicates to the system, for each constraint in the problem, the name given to the constraint, the sign and the RHS value for the constraint.

c. The specification list of the CONSTRAINTS statement may be continued on as many cards as are necessary in which case every continuation card must have an * in column 1.

d. The CONSTRAINTS statement is immediately followed by the activity matrix (a_{ij}) in the form specified by the FORMAT statement.

C. *The data-modification statements*

1. The DELETE statement

a. The general form of the DELETE statement is

```
DELETE        name, . . . . . . }
              NO                }
```

where *name* is the name of a variable or constraint, as specified by the VARIABLES or CONSTRAINT statement

b. The DELETE statement causes the variables and/or constraints named in the specification list to be deleted from the original set of variables and/or constraints as defined by the data-definition statements.

c. The DELETE statement always operates on the original set of variables and/or constraints. Once invoked, it remains in effect throughout subsequent subtasks, if any, until superseded by another DELETE statement. The statement DELETE NO restores the original set of variables and constraints.

2. The MODIFY statement

a. The general for of the MODIFY statement is

```
MODIFY        list 1/list 2/list 3 }
              NO                   }
```

where *list 1* is of the form *vname* (xxx), . . .
list 2 is of the form *cname* sign (xxx), . . .
list 3 is of the form *cname vname* (xxx), . . .

b. Depending on which type of list appears in the specification list of the MODIFY statement, prices, signs, RHS values and/ or activity coefficients may be modified.

c. The MODIFY statement always operates on the original data as defined by the data-definition statements. Once invoked, it remains in effect throughout subsequent subtasks, if any, until superceded by another MODIFY statement. The statement MODIFY NO restores the original data.

3. The SCALEFACTOR statement

a. The general form of the SCALEFACTOR statement is

```
SCALEFACTOR  xxx
```

where xxx is a positive, non-zero constant

b. The SCALEFACTOR statement multiplies all RHS values by xxx.

D. *The specification statements*

1. The ECHOPRINT statement

a. The general form of the ECHOPRINT statement is

$$\text{ECHOPRINT}\qquad\left\{\begin{matrix}\text{PRIMAL}\\ \text{DUAL}\\ \text{NO}\end{matrix}\right\}$$

b. The ECHOPRINT statement produces a printout of the input data in two separate sections: the objective function section and the constraints section.

c. If PRIMAL is specified, then the data as defined by the data-definition statements will be printed. If DUAL is specified, the corresponding dual problem will be generated (subject to the restriction that there be no equation-type primal constraints) and then printed.

d. The ECHOPRINT statement is optional; the default condition is no printout of the input data.

2. The PRINTFORMAT statement

   a. The general form of the PRINTFORMAT statement is

$$\text{PRINTFORMAT} \quad \begin{Bmatrix} \text{F} \\ \text{E} \end{Bmatrix}$$

   b. The PRINTFORMAT statement allows the user to choose between the F and E format codes in the printing of all numerical values.

   c. The PRINTFORMAT statement is optional; the default option is the E format code.

3. The METHOD statement

   a. The general form of the METHOD statement is

$$\text{METHOD} \begin{Bmatrix} \text{BIG-M} \\ \text{TWO-PHASE} \end{Bmatrix}$$

   b. The METHOD statement gives the user a choice between two implementations of the simplex algorithm: the big-M or the two-phase method.

   c. The METHOD statement is optional; the default option is the two-phase method.

4. The TABLEAUX statement

   a. The general form of the TABLEAUX statement is

$$\text{TABLEAUX} \begin{Bmatrix} \text{FIRST} \\ \text{LAST} \\ \text{FIRST, LAST} \\ \text{ALL} \\ \text{NO} \end{Bmatrix} \quad [*]$$

   b. Computational results may be printed in tableau form. Every iteration of the simplex algorithm results in a new tableau, which the user may or may not wish to be printed. The keywords within the braces are the options allowed the user; e.g., FIRST means "print the first tableau only", and so on.

c. The * is optional. It is meaningful only if TWO-PHASE is specified by the METHOD statement. In this case, if * is coded, then the corresponding first phase tableaux (first, last, etc.) will also be printed; otherwise, only second phase tableaux will be printed.

d. The TABLEAUX statement is optional; the default option is no printout of any tableaux.

5. The SUMMARY statement

a. The general form of the SUMMARY statement is

```
SUMMARY      [NO]
```

b. The SUMMARY statement produces an iteration log. Specifically, it prints, for every iteration of the simplex algorithm, the names of the outgoing and incoming vectors, the $z_j - c_j$ for the incoming vector and the functional value.

c. The SUMMARY statement is optional; the default option is no printout of iteration log.

6. The ITERATIONS statement

a. The general form of the ITERATIONS statement is

```
ITERATIONS    xxx
```

where xxx is a non-negative integer constant

b. The ITERATIONS statement allows the user to specify an upper limit on the number of iterations of the simplex method to be performed. (It is pertinent to state. at this point that LPL user Charne's perturbation technique to prevent cycling in case of degeneracy; hence, the user need not fear that the iterations will never cease due to cycling.)

c. The ITERATIONS statement is optional; the default option is : perform as many iterations as needed until unboundedness or infeasibility is detected, or an optimal feasible solu tion is obtained.

7. The RANGES statement

   a. The general form of the RANGES statement is

$$\text{RANGES} \quad \left\{ \begin{matrix} \text{ALL} \\ \text{selection list} \\ \text{NO} \end{matrix} \right\}$$

$$\text{where selection list} = \left[\begin{matrix} \text{BASIC} \\ \text{NONBASIC} \end{matrix} \right] \left[\begin{matrix} \text{OFC} \\ \text{RHS} \end{matrix} \right] \quad [*]$$

   b. The RANGES statement allows the user to perform a sensitivity analysis on an optimal solution. Specifically, he can determine the range over which a given objective function coefficient or right-hand-side coefficient may be varied, while holding all other values constant, before the current optimal basis changes.

   c. The RANGES statement is optional; the default option is no sensitivity analysis performed on OFC and RHS.

8. The RENAME statement

   a. The general form of the RENAME statement is

$$\text{RENAME} \qquad \text{label}$$

   b. The RENAME statement allows the user to change the TASKNAME — specified label, which is printed as heading on every output page, with the label declared in the RENAME statement.

E. *The MAXIMIZE or MINIMIZE Statement*

   a. The MAXIMIZE or MINIMIZE statement is alone in its class. Its general form is

$$\left\{ \begin{matrix} \text{MAXIMIZE} \\ \text{MINIMIZE} \end{matrix} \right\} \left\{ \begin{matrix} \text{PRIMAL} \\ \text{DUAL} \end{matrix} \right\} \quad [\text{ofn}]$$

where *ofn* is any string of at most 12 alphameric characters. The special characters + - . may also be used. *ofn* is the name given to the objective function.

b. The MAXIMIZE or MINIMIZE statement is the logically last statement in a task or subtask.

c. If PRIMAL is specified, the primal problem, as defined by the data-definition statements, will be solved. If DUAL is specified, then the dual problem is generated and solved, subject however to the restriction that there be no equation-type primal constraints.

F. *The run statements*

1. The NOLIST statement

   a. The general form of the NOLIST statement is

```
NOLIST
```

   b. LPL automatically produces a listing of the source statements. If for some reason the user wants to suppress this listing, then he should code.

```
NOLIST
TASKNAME...
More LPL statements
TASKEND
STOPRUN
```

2. The STOPRUN statement

   a. The general form of the STOPRUN statement is

```
STOPRUN
```

   b. The STOPRUN statement is the physically last statement in an LPL program. It terminates an LPL run, which may consist of one or more LPL tasks.

3. The : comment statement

   a. The general form of the : comment statement is

```
:   comment
```

where the colon must be in column 1 of the card and *comment* is any string of characters.

b. The follówing example illustrates the use of this statement (only control words are shown)

```
:   THIS IS A SAMPLE LPL PROGRAM
TASKNAME. . .
More LPL statements
MAXIMIZE. . .
:   MODIFY THE PROBLEM DATA AND SOLVE A NEW
MODIFY . . .
MAXIMIZE . . .
TASKEND
STOPRUN
```

After this brief description of the various LPL statements, a question that naturally arises is : What happens if a statement is not correctly coded? Well LPL will issue an error message. Actually, the LPL interpreter issues two types of error messages — warnings and diagnostics. Consider, for example, the following LPL program.

```
TASKNAME
VARIABLES VI (10.75), V2 (12.65), V3 (-11.14)
FORMAT FREE LIST #∈*;!
CONSTRAINTS c1 < 20.65, c2/10.00
    (more LPL statements)
TASKEND
.
.
.
.
.
```

The first and second statements are correctly coded. The FORMAT statement, on the other hand, contains superfluous information after the keyword LIST. Since the correct information would already have been extracted before the #∈*;! is encountered the LPL interpreter simply issues a warning of the form.

```
*****  CARD IMAGE       FORMAT FREE LIST #∈*:!
*******WARNING          CARD CONTAINS SUPERFLUOUS
                        INFORMATION AT OR TO THE
                        RIGHT OF COLUMN xxx
```

and proceeds to interpret the CONSTRAINTS statement. However, since *sign* in the specification list of the CONSTRAINTS statement is < or = >, the / following c2 cannot be properly interpreted. In this case, LPL issues a diagnostic of the form.

```
*****CARD IMAGE        CONSTRAINTS c1 < 20.66, c2/10.00
*****DIAGNOSTIC        INVALID FIELD AT OR TO THE
                       RIGHT OF COLUMN xxx
```

and the task is aborted. The next task, if any, is then processed. Otherwise the run is terminated.

In addition to errors in syntax, LPL also detects logical errors. For example, if the statement

```
DELETE v1, v2, v3
```

is coded into the sample program above, then LPL will print the message

```
*****CARD IMAGE        DELETE v1, v2, v3
*****DIAGNOSTIC        THE * DELETE * STATEMENT
                       MAY NOT DELETE ALL THE
                       VARIABLES
```

upon encountering this statement, and the task is terminated.

A description of all the error messages issued by LPL will be too lengthy to include here. It should suffice to state of this point that, as a general rule, the image of the card in error, or the TASKNAME or RENAME label, is printed along with the message. This makes debugging relatively easy.

LPL: Its Features as an Interpretive System

LPL, as an interpretive system, consists of 50 FORTRAN subprograms and 12 assembler routines operating under a FORTRAN control program. Seven of the 12 assembler routines are generated by two macroinstructions of an executive system generator called STAPLES*. The important tasks of command decoding and extraction of numeric and non-numeric data inputted in free-field fashion are done with relative ease through FORTRAN explicit and implicit calls to these STAPLES-generated modules. The STAPLES macroinstructions which LPL utilize are:

(1) Macroinstructions CMD — generates tables of keywords and entry point addresses to processing programs

*STAPLES (Structured Adaptation of Problem-Oriented Languages for Engineering Systems) was developed by Dr. Salvador F. Reyes of the Department of Civil Engineering, College of Engineering, U.P. It is currently implemented on the IBM/360-256 system of the U.P. Computer Center.

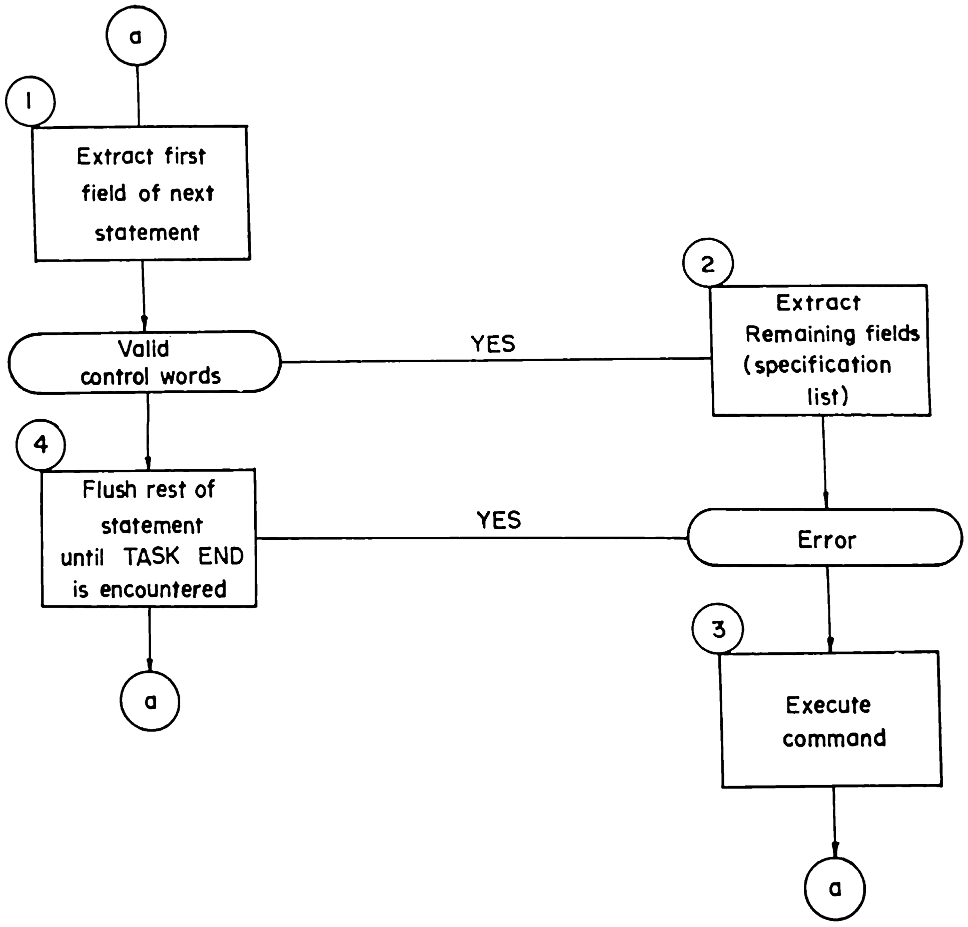
a
1
Extract first field of next statement
Valid control words
YES
2
Extract Remaining fields (specification list)
4
Flush rest of statement until TASK END is encountered
YES
Error
a
3
Execute command
a

FIG. 3 FLOW DIAGRAM OF STATEMENT INTERPRETATION

(2) Macroinstruction PRV — generates subprograms for performing miscellaneous tasks not readily accomplished by FORTRAN coding, e.g., referencing a FORTRAN variable by its address

It will be recalled that an LPL program consists of a sequence of statements punched on 80-clumn cards. Most of these statements are completely contained in a single card, while others may be continued onto the next and succeeding cards. In any case, LPL sequentially processes these statements, i.e., it reads, decodes and executes the statements as they are encountered. Figure 3 below illustrates by means of a flow diagram the mechanics of interpreting LPL statements. For reference, the boxes are numbered (1) through (4).

In the previous section, it was mentioned that LPL statements, in general, consists of two parts: a control word and a specification list (see Fig. 2). In box (1), Fig. 3, the first field containing the control word is extracted by the LPL interpreter (a FORTRAN program) and stored in a working vector. This is then passed to the assembler routine generated by macroinstruction CMD, which determines whether the control word is valid or not. If it is valid, the routine returns the entry point address of the appropriate subprogram which in turn will process the rest of the statement (the specification list). In box (2), the appropriate subprogram takes over the decoding, checking as it decodes for errors. In box (3), the phrase "execute command" stands for a variety of actions which the system may take depending on the command decoded. For example, it may mean simply turning on or off certain system switches, transforming data in memory, invoking computational routines, and the like. If at any point in the interpretation process an error is detected, then the task is aborted (box (4)), and the next task, if any, is processed.

Some of the details of this general scheme of statement interpretation will now be discussed in the remainder of this section.

A. *Command Decoding*

Command decoding, as employed in LPL, involves two steps, namely: (1) determining whether the control word is valid or not (e.g., PRINTFORMAT is valid but PRINT FORMAT is not); and (2) on the condition that the control word is valid, transfering control to the appropriate routine which will decode and execute the rest of the statement. STAPLES macroinstruction CMD provides precisely these two capabilities, and in a manner that can only be described as elegant. To see how LPL utilizes the CMD macroinstruction, consider the following segments of IBM/360 assembler and FORTRAN code.

```
            START       O
LPLSM       CSECT
            DS          D
            .
            .
            .
CMD *, TN  = LPLCT, ST = LPLST, KL = 12, PN = LPLØ2
CMD 1,  STOPRUN
CMD 2,  TASKEND
CMD 3,  TASKNAME, LPLØ3
CMD 4,  VARIABLES, LPLØ4
.
.
.
CMD   19,   MINIMIZE, LPL18
CMD
             .
             .
             .
C . . . . . . . . . . *LPL* CONTROL PROGRAM
      COMMON  B  (101,  203),  KARD  (20),  K  (20),  INFOR  (30),
         VNAME (400), CNAME (200)
      COMMON IN (100), IS (100), PRICE (201)
      COMMON  IPTR, IERR, NVAR, NCON, IREC, NVT, NTP, MNTP,
         NTAB, ITER, LOG, KODE, SF
      EXTERNAL LPLSW
      .
      .
      .
C . . . . . . . . . INTERPRET NEXT STATEMENT
      3 CALL LPLØ(LPLSW)
C . . . . . . . . . EXECUTE STATEMENT
      CALL LPLSW
C . . . . . . . . . RETURN TO SYSTEM
      GO TO 3
      END
      .
      .
      .
C . . . . . . . . . *LPL* INTERPRETER
      SUBROUTINE LPLØ1 (LPLSW)
      COMMON B (101, 203), KARD (20), K (20), INFOR (30),
         VNAME (400), CNAME (200)
```

```
C.......... GET STATEMENT KEYWORD
    9 READ (10) KARD
      IPTR   =  1
      K(1)   =  1077952576
      K(2)   =  1077952576
      K(3)   =  1077952576
      IF (KPKFLD (K, KARD, IPTR, 80) 10, 15, 11
   10 IF (KSTRC1 (KARD)) 15, 15, 17
C.......... INTERPRETE KEYWORD
   11 CALL LPLØ2 (K, LPLCT, LPLSW, LPLST, KOMAND)
      IF (KOMAND -1) 15, 21, 13
   13 IF (KOMAND -2) 7, 7, 19
C ........... INTERPRETATION FAILED : UNIDENTIFIED KEY-
    WORD
   15 WRITE (3, 100) KARD
      .
      .
      .
C ........... INTERPRETATION SUCCESSFUL : RETURN TO
    CONTROL PROGRAM
   19 RETURN
      END
```

Consider first the sequence of CMD statements in the given segment of assembler code. The first statement indicates, among other things, that the routine generated by the CMD macroinstruction will be called LPLØ2. Next comes a series of statements of the form,

```
CMD n, cw, sn
```

where n is simply a sequence number

cw is a control word which identifies an LPL statement
sn is the name of the subprogram which will process the statement

For example,

```
CMD 3, TASKNAME, LPLØ3
```

means that the keyword TASKNAME is assigned the sequence number 3 and that the TASKNAME statement (strictly, the specification list of the TASKNAME statement) will be processed by subprogram LPLØ3.

The process of command decoding is initiated in statement 3 of the LPL control program (see given segment of FORTRAN code), in which subroutine LPLØ 1, the LPL interpreter, is called. In statement 9 of subroutine LPLØ 1, the next card is read and its image is stored in the 20-word array KARD. After initializing IPTR (which indicates where field extraction begins) to 1 and the receiving field (the first three words of the 20-word array K) to blank, function KPKFLD is invoked. This subprogram extracts the first field in KARD and installs this in K. For example, if the statement that was read in statement 9 of subroutine LPLØ 1 is

```
TASKNAME     SAMPLE *LPL* RUN
```

then the first three words of K will contain

```
K(1)   =  'TASK'
K(2)   =  'NAME'
K(3)   =  '    '
```

In statement 11, K is passed to the CMD-generated routine LPLØ 2. Pursuing the example above, LPLØ 2 now returns in KOMAND the sequence number, 3, of the keyword TASKNAME and in LPLSW the entry point address of subprogram LPLØ 3. The interpretation being successful, control returns (statement 19 of LPLØ 1) to the statement

```
CALL      LPLSW
```

in the control program. In effect, the control program calls LPLØ 3, which then interprets the rest of the statement.

B. *Free-field Input of Numeric and Non-numeric Data*

One convenient feature of LPL as a language is the free-field input of both numeric and non-numeric data. This means that there is no fixed column assignment for any entity appearing in a statement. For example, the two statements shown below are equivalent, i.e., they will be interpreted in exactly the same way.

```
VARIABLES  VARBO1 (10.75 VARBO2 (27.86) VARBO3 (9.67)
```

This capability is provided by modules generated by macroinstruction PRV. Two modules are particularly useful in this respect: KPKFLD and KNVRTR. Consider the following program segment from subroutine LPLØ 4, which processes the specification list of the VARIABLES statement:

```
C.......... *VARIABLES* statement
      SUBROUTINE LPLØ4
      COMMON B (101, 203), KARD (20), K(20), INFO (30),
          VNAME (400), CNAME (200)
      COMMON IN (100), IS (100), PRICE (201)
C.......... EXTRACT A FIELD
    7  IPTR = 1 POINT
       K(1) = 1077952576
       K(2) = 1077952576
       K(3) = 1077952576
       LEN  = KPKFLD (K, KARD, 1 POINT, 8Ø )
       .
       .
       .
C.......... VARIABLE NAME

       .
       .
       .
   15  IF (MATCH (K, VNAME, 8, NVAR, 5) 103, 17, 103
   17  NVAR =   NVAR  +  1
       GO TO 7
C.......... PRICE
       .
       .
       .
       IPT =  1
       IF (KNVFTR (PRICE (NVAR), 2, K, IPT, 12)) 101, 101, 7
       .
       .
       .
```

Assume that KARD contains

```
     Col. 3         12       18
        ↓           ↓        ↓
        VARIABLES   x1(25.14)   x2(15)
```

Upon entry into subroutine LPLØ 4, field extraction begins at columr 12, which is the value of 1 POINT in the statement

```
LEN  = KPKFLD (K, KARD, 1 POINT, 8Ø )
```

The effect of the above statement is to extract from KARD and install in K the string x1 (a variable name), after which the value of 1 POINT is updated to 18.

In statement 15, the extracted variables name is appended to the name table VNAME and the count of variables, NVAR„ increased by 1. Then the next field is processed (GO TO 7).

The coefficient 25.41 is extracted by KPKFLD as a string, in the same way that x1 and x2 (are extracted as strings). If '25.14' is to be used in a calculation, as it will be in the simplex computations, then it must be converted into the appropriate binary form. This conversion is performed by function KNVRTR. The statements

```
.
.
.
IPT    =  1
IF 9KNVRTR 9 PRICE 9NVAR0, 2, K, IPT, 12) 101, 101,7
.
.
.
```

in the program segment shown above perform precisely this task. After the conversion, the string '25.14' stored in K is moved as the real constant 25.14 into PRICE(NVAR).

Conclusion

LPL is designed primarily for class use. The availability of a problem-oriented language such as LPL should relieve the student of the burden of having to carry out tedious, repetitive calculations. With the computer easily instructed to perform the simplex computations, the student should have ample opportunity to formulate more realistic LP problems, to try different alternatives, and to gain more insight into the problem as he interprets the computer printouts.

References

1. Gass, S., "Linear Programming," McGraw-Hill, 1975.

2. Hadley, G., "Linear Programming," Addison-Wesley, 1963.

3. Mathematical Programming System/360:Version 2, Linear and Separable Programming — User's Manual (Form 360A-CO-14x)

4. Reyes, Salvador F., "STAPLES — Structured Adaptation of Problem-Oriented Languages for Engineering Systems", (unpublished paper).