Abstract

This paper contains three parts:

1. A set of tutor models, models that guide you through several aspects of LPL modeling from a simple production model to more complex ones. The tutor examples start with a simple small production model. Successively, more features are added to the model to illustrate the syntax of the language LPL. Be aware that some models need Excel or a database to be installed before running correctly. Otherwise all can be run on the Internet without installing LPL locally.

2. A set of “learn” examples which are model fragments explaining various aspects or functions of the LPL language, they are also linked in the reference manual to explain various concepts of the language.

3. Some drawing examples explaining the drawing library of LPL. LPL contains a library that can generate SVG graphs. They can be interpreted by any browser directly.

This text was automatically generated using LPL’s own documentation facility, and illustrates itself an important feature of LPL. Going through the examples together with loading and running them using lplw.exe or the Internet interface gives you a first overview of the capacity of the modeling system LPL. The reader is asked not only to read this tutorial but also to load and run all models. It is the most efficient way to go into the modeling language. Enjoy it!
## Contents

1. A Simple Production Model (*tutor01*)  
2. Names and Comments (*tutor02*)  
3. Using Indices I (*tutor03*)  
4. Using Indices II (*tutor04*)  
5. Using Indices III (*tutor05*)  
6. Data Include Files (*tutor06*)  
7. Reading text files (*tutor07*)  
8. Reading data from Excel I (*tutor07a*)  
9. Reading data from Excel II (*tutor07a1*)  
10. Reading data from Database (*tutor07b*)  
11. Writing to text files (output locally only) (*tutor08*)  
12. Writing data to Excel (*tutor08a1*)  
13. Writing to databases (output locally only) (*tutor08b*)  
14. Creating a Report (report locally only) (*tutor08c*)  
15. Writing With Formatted Masks (*tutor08d*)  
16. Write-Format Examples (*tutor08e*)  
17. Submodels (output locally only) (*tutor09*)  
18. Sparse Tables (*tutor10*)  
19. Predefined Functions (*tutor11*)  
20. Index Operators (*tutor12*)  
21. Expression Evaluation (*tutor13*)  
22. Goal Programming (*tutor14*)  
23. Loop Programming (*tutor15*)  
24. Logical Constraints (*tutor16*)  
25. Some Basic Expressions / Writes (*learn01*)  
26. Basic Indexed Expressions/Tables (*learn01a*)
27 The Addm function (learn01b) 67
28 Some Math and Boolean Functions (learn02) 68
29 Logical Operators (learn02a) 69
30 The within and in Operators (learn03) 70
31 Some String Compare Operations (learn04) 71
32 Index options, and wrap around function (learn05) 72
33 Sort function (learn06) 73
34 Date/Time Type (learn07) 74
35 Documenting Models (learn08) 75
36 Call Submodel within Solver (Gurobi) (learn10) 77
37 Reading Relations (learn12) 78
38 Multiple bounds of variables (learn14) 79
39 Submodels and Encapsulation (learn15) 80
40 Again, some non-trivial relations (learn16) 82
41 Expressions and Constraints (learn17) 83
42 Get Value Function (learn20) 84
43 create a SQL script, test sparcity (locally only) (learn21) 85
44 Get Attr Function (two parameters) (learn22) 86
45 Get Attr Function (one parameter) (learn22a) 87
46 GetName Function (learn23) 88
47 Get Params Function (learn24) 89
48 Split Function (learn25) 90
49 String Functions (learn26) 91
50 String Operations and Format (learn26a) 92
51 Functions Next Focus, Next Position (learn27) 93
52 Multiple Snapshots (learn29) 94
53 Show Graph.Component (learn30) 95
54 Show Graph.Mincut function (learn31) 96
55 Show Graph.MStree (Minimal Spanning Tree) (learn32) 97
56 Read Multiple Snapshots (learn33) 98
57 Freeze function (learn34) 100
58 function En(), El() (learn35) 101
59 Show Graph.Bfs (learn36) 102
60 Show Graph.Path (shortest path) (learn37) 103
61 Sl function (goal programming) (learn39) 104
62 Parameterized Calling of Submodels (learn40) 106
63 A Model with Parameters I (learn41) 108
64 A Model with Parameters II (learn42) 110
65 Pivot Table (learn43) 111
66 Colored Report (learn44) 112
67 Circular Time Lag Operator (learn45) 113
68 Function xP (learn46) 114
69 Draw Lines/Opacity (xDrawAll) 115
70 Use arc path (xDrawArcPath) 117
71 Drawing Circles (xDrawCircle) 118
72 A Quadratic Constraint (xDrawCircle1) 119
73 Show Color Numbers in LPL (xDrawColors) 120
74 Draw Filters (xDrawFilters) 121
75 Show radial gradient (xDrawGrad0) 122
76 Show linear gradient (xDrawGrad1) 123
77 Show figures with gradient (xDrawGrad2) 124
78 Check Points inside a Polygon (xDrawInside) 125
79 Check Two Segments intersect (xDrawIntersect) 126
80 Drawing Lines (xDrawLine) 127
81 Draw a path (xDrawPath) 128
82 Draw a pattern (xDrawPattern) 129
83 Load Picture from File (xDrawPict) 130
84 Some drawing functions (xDrawPict1) 131
85 Puzzle 64=65 (xDrawPuzzle) 132
86 Draw rectangles (xDrawRect) 135
87 Draw.Scale Function (xDrawScale) 137
88 Drawing Text (xDrawText) 138
89 Drawing a TextPath (xDrawTextPath) 139
90 Draw a Text on a Path (xDrawTPath) 140
91 Show transformations (xDrawTrans1) 141
92 Draw Penrose Triangle (xDrawxPenrose) 142
93 Draw Sierpinski Triangles (xDrawxSierpinski) 143
94 Draw a colored spiral (xDrawxSpiral) 144
95 Drawing Library I (xDrawXY) 145
List of Figures

1. The Feasible Space (grey) ................................................. 8
2. Data in Excel Sheet ...................................................... 27
3. Data in Excel Sheet (“free style”) ................................. 29
4. Data in the database .................................................... 32
5. Table HoursTable1 in the database ................................. 33
6. Data in Excel Sheet (“free style”) ................................. 38
7. The generated Database ................................................ 39
8. The generated Database ................................................ 42
10. Output ................................................................. 116
11. Output ................................................................. 117
12. Output ................................................................. 118
13. The Solution ............................................................ 119
14. Output ................................................................. 120
15. Output ................................................................. 121
16. Output ................................................................. 122
17. Output ................................................................. 123
18. Output ................................................................. 124
19. Output ................................................................. 125
20. Output ................................................................. 126
21. Output ................................................................. 127
22. Output ................................................................. 128
23. Output ................................................................. 129
24. Output ................................................................. 130
25. Output ................................................................. 131
26. $64 \div 65$ ............................................................. 132
27. $169 \div 168$ ............................................................ 134
28. $25 \div 24$ ............................................................... 134
29. Output ................................................................. 136
30. Output ................................................................. 137
31. Output ................................................................. 138
32. Output ................................................................. 139
33. Output ................................................................. 140
34. Output ................................................................. 141
35. Output ................................................................. 142
36. Output ................................................................. 143
37. Output ................................................................. 144
38. Output ................................................................. 145
1 A Simple Production Model (tutor01)

Problem: [This part of the documentation is intended to briefly explain the problem.]
A firm produces two type of robots called Marie and Jules. Three production steps must be carried out:

1. Production of the components: It takes 5 hours for each robot Marie and Jules, with a total capacity of 350 hours per week,

2. Mounting (capacity = 480): It takes 4 hours for mounting one robot of type Marie and 8 hours for one robot of type Jules,

3. Testing (capacity = 300): It takes 6 hours for one Marie and 2 hours for one Jules.

The revenue for Marie is $300 and for Jules $200. There are already 20 robots of type Marie and 15 Jules ordered. How many robots of each type can be produced per week, if the firm wants to maximize the selling revenue (costs are not considered in this simple problem)? (1)

Modeling Steps
[[This part of the documentation explains how the problem can be translated into a mathematical model.]]

1. Let’s introduce two variables: x and y for the quantity (per week) of the two types of robots to be produced.

2. The components-step has a capacity of 350 hours per week. A unit of robot Marie takes 5 hours, a robot Jules also takes 5 hours. Hence we have (the component capacity per week):

   \[5x + 5y \leq 350\]

3. The mounting-step has a capacity of 480 hours per week. A robot Marie takes 4 hours, a robot Jules takes 8 hours. Therefore:

   \[4x + 8y \leq 480\]

4. The testing-step has a capacity of 300 hours per week. A robot Marie takes 6 hours, a robot Jules takes 2 hours. Therefore:

   \[6x + 2y \leq 300\]

5. At least 20 of type Marie and 15 of type Jules must be produced, hence:

   \[x \geq 20, \quad \text{and} \quad x \geq 15\]

6. Maximizing the revenue is:

   \[
   \text{max } 300x + 200y
   \]
The complete problem to solve is then as follows:

\[
\begin{align*}
\text{max} & \quad 300x + 200y \\
\text{subject to} & \quad 5x + 5y \leq 350 \\
& \quad 4x + 8y \leq 300 \\
& \quad 6x + 2y \leq 300 \\
& \quad x \geq 20 \quad y \geq 15
\end{align*}
\]

**Further Comments:** In a graphical way, the model represents the solution space as shown in Figure 1. That is, every \((x, y)\)-point within the gray polygon represents a possible (feasible) production that fulfills all constraints. All other points in the two-dimensional space violate at least one of the constraints.

![Figure 1: The Feasible Space (grey)](image)

**Listing 1: The Complete Model in LPL [1]**

```lpl
model TUTOR01 "A Simple Production Model";
variable x; y;
constraint
  Component: 5*x + 5*y <= 350;
  Mounting: 4*x + 8*y <= 480;
  Testing: 6*x + 2*y <= 300;
  Order1: x >= 20;
  Order2: y >= 15;
maximize revenue: 300*x + 200*y;
Writep(x,y);
end
```

**LPL Modeling Steps**

8
[The goal of this tutor is mainly to explain the syntax of LPL. It is done in this text block.] A LPL coded mathematical model [1] is very close to the usual mathematical notation. However, we have some additional elements to code:

1. Each model coded in LPL begins with a keyword model and ends with end. (Keywords are all in lowercase.)

2. The code consists of a sequence of entity declarations and statements.

3. Each entity declaration begins with a keyword and ends with a semicolon.

4. There are four kinds of entities in this model: model, variable, constraint, and maximize. There is no need to repeat the keywords variable or constraint for consecutive entities of the same type. Therefore variable introduces a list of two variables.

5. constraint introduces the model constraints. Each constraint then begins with a name. Then follows a colon and the constraint expression.

6. The objective function, with the name revenue, begins with maximize.

7. Finally, we want to output the solution with the function Writep.

Solution: [[This part of the documentation is intended to give a short comment about the solution of the model.]] The optimal solution is \( x = 40 \) and \( y = 30 \), which can be verified by Figure 1. It means that neither the quantity of \( x \) nor the quantity of \( y \) can be increased without violating one of the capacity constraint. On the other hand, a reduction of one of the quantities is ‘suboptimal’: it reduces the revenue, and hence it is no longer maximized.

Question (Answer see 1)

1. Verify the solution of this model by running it.

2. What happens if we extend the Mounting capacity from 480 to 500?

3. We must produce 45 unit of Jules instead of 15. How must the model be changed?

4. Besides the three production steps, there is a fourth task: “storage”. The company has a storage capacity of 300 storage units and each Marie uses 7 and each Jules uses 2 storage units. Add this requirement to the model and solve the problem again. Does the new task influence the result?

Answer (Question see 1)

1. Locally, run lplw.exe. Choose Menu “File/Open” and open the file tutor01.lpl. Now choose Menu "Run/Run Model". Click the tab 'TABLE', and then click on the red node 'x' in the left part in the LPL application. The value 40 is displayed. Now click on the node 'y', the value 30 is displayed. OR: Click the red link called “tutor01” at the beginning of this section. The model opens in a Web browser. Click “Run and Solve”.

9
2. Nothing happens! The solution is still \(\{x = 40, y = 30\}\). This can be seen immediately from Figure 1: The Mounting capacity is not the limiting resource. So, extending it, does not change anything. To verify this, change 480 to 500 in the model and run it. Then look at the values of the variables again.

3. Change the constraint Order2 to \(y \geq 45\). Then run the model again. The solution indicates that 45 units of Jules are produced, but the number of Marie drops to 25, and the overall revenue also drops to 16500. This can be seen by clicking the blue node revenue while 'TABLE' is the active tab.

4. The storage requirement can be added to the model by defining an additional constraint, which is

\[
\text{constraint Storage: } 7x + 2y \leq 300;
\]

The optimal solution is: 32 of Marie and 38 of Jules. Hence, the storage capacity limits the production substantially.
2 Names and Comments (tutor02)

Problem: This is exactly the same model as tutor01.lpl. Comments are added and some simple expressions are used.

Listing 2: The Complete Model in LPL [1]

```lpl
model TUTOR02 "Names and Comments";
    /* The variables are declared */
    variable Marie,x "Number of robots Marie";
    Jules,y "Number of robots Jules";
    /* The constraints are listed */
    constraint
        Component: 5*x + 5*y <= 300+50 "Capacity of comp per week";
        Mounting: 4*x + 8*y <= 500-20 "Capacity of mount per week";
        Testing: 6*x + 2*y <= 30*10 "Capacity of test per week";
        Order1: x >= 200/10 "Already ordered of Maries";
        Order2: y >= 4^1.9534 "Already ordered of Jules";
        // This function is maximized
        maximize revenue: 300*x + 200*y;
        —— Finally, three data tables are written to the NOM-file
        Writep(revenue, x, y);
end
```

LPL Modeling Steps

One can add comments, blanks or linefeeds everywhere between tokens (words). Furthermore,

1. An entity can have more than one name: Variable Marie, e.g., has two names: Marie and x. This will be especially useful for indexes (see later). A second name is introduced by adding it after the first name separated by a comma.

2. Multiline comments must be within /* and */. They can be nested.

3. Short one-line comments begin with a double-dash (--) or with //.. They end with an end-of-line.

4. Expressions, like 300 + 50, can be used.

5. The 5 operators used here are: + (add), - (minus), * (times), / (divide) and ^ (power).

6. maximize calls a default solver library or program and reads the results back into the LPL.

7. Write writes the results to the so called NOM-file (here the filename is: tutor02.nom). In lplw.exe this file is loaded automatically after a run and shown in a new tab.

Question (Answer see 2)

1. What happens if one adds two dashes just before the word Testing?

2. Outcomment the line of Jules,y .... What happens?
3. What happens if we change the name Marie to Mary?

**Answer** (Question see 2)

1. The constraint Testing is commented and does not taking any effect when resolving the problem. That is, the solution will be now: \( \{x = 55, y = 15\} \).

2. Running the model will stop at the first occurrence of \( y \) and an error message is output at the status line: *Error: 516 Unknown identifier.*

3. Nothing! The name is not used elsewhere in the model.
3 Using Indices I (tutor03)

Problem: [This model now introduces the very fundamental concept of index.] Suppose, we have 10 different types of robots (not just 2 like in the previous models tutor01 and tutor02. A way to deal with this is to introduce 10 different variables. However there is a more economical way, that is, using indexes. In mathematical notation, one can write this as follows:

\[ x_i, \text{ with } i \in I = \{1\ldots10\} \]

In LPL syntax, a very similar notation is used:

```lpl
set I := 1..10;
variable x{i in I};
```

In the same way, in mathematics, one used the summation notation to add all variables. The “mounting robots” capacity constraint can be formulated as:

\[ \sum_{i \in I} H_M_i \cdot x_i \leq 4800 \]

where \(H_M_i\) is the time spent to mount a single robot type \(i\), 4800 being the total capacity for this production step. In LPL, again, this is formulated close to this notation as follows (note that every constraint has a name in LPL, here Mounting, to identify it within in the LPL code):

Listing 3: The Complete Model in LPL [1]

```lpl
model TUTOR03 "Using Indices I";
set I := 1..10 "A set with 10 elements";
integer variable x{i in I} "The number of different type of robots";
parameter HC{i in I} := [ 5 5 4 5 6 5 7 8 4 7 ] "Component time";
HM{i in I} := [ 4 8 5 6 4 8 7 6 5 3 ] "Mounting time";
HT{i in I} := [ 6 2 4 6 3 4 5 2 5 3 ] "Testing time";
Ordered{i in I} := [ 20 15 7 6 5 8 9 8 7 5 ] "Quantity ordered";
Price{i in I} := [300 200 100 50 50 100 200 100 400 200 ];
constraint Component: sum{i in I} HC[i] * x[i] <= 3500 "Component building";
Mounting: sum{i in I} HM[i] * x[i] <= 4800 "Mounting robots";
Testing: sum{i in I} HT[i] * x[i] <= 3000 "Testing robots";
Order{i in I}: x[i] >= Ordered[i] "Ordered";
maximize revenue: sum{i in I} Price[i] * x[i] "Maximize the revenue";
Writep(revenue, x, HC, HM, HT);
end
```

LPL Modeling Steps

The new elements in this model are set and the index-operator sum.

1. The set entity declares an index-set called \(I\). The index-set \(I\) has 10 elements.

\[ I = \{1,\ldots,n\} \quad (n = 10) \]

2. variable introduces a variable list called \(x\), which is indexed over \(i \in I\). \(i\) is called an index. This declares the 10 variables: \(x[1] \ldots x[10]\).
3. **parameter** introduces five data lists, all indexed over $I$. (For example, $HC[3]$ (which is 4, the third data in the list) says how many hours it take to manufacture the components for the third robot type 3.)

4. The data list is directly assigned to the parameters. Hence, we have $Price[1]=300$ and $Price[6]=100$, for example. These data are data vectors.

5. Indexed items can be summed up with the **sum** operator. Hence,

   \[ HC[1] \cdot x[1] + \ldots + HC[10] \cdot x[10] \]

   is written as: $\sum_{i \in I} HC[i] \cdot x[i]$. This is LPL’s notation for:

   \[ \sum_{i \in \{1\ldots10\}} HC_i \cdot x_i \]

6. Much like the summation through the **sum** operator, whole constraint-classes can be written as indexed constraints. For example:

   \[ Order(i \in I): x[i] \geq Ordered[i]; \]

   declares 10 constraints:

   \begin{align*}
   x[1] & \geq Ordered[1], \\
   x[2] & \geq Ordered[2], \\
   & \ldots
   \end{align*}

   In mathematical notation, we would write this as follows:

   \[ x_i \geq Ordered_i, \quad \forall i \in I = \{1\ldots10\} \]

7. A **Writep** instruction can be used to write the results to the NOM-file which is the result file by default. In this case, resulting tables are written to the file `tutor03.nom`.

**Question** (Answer see 3)

1. Replace $1..10$ by $1..11$ in the set definition of $I$. What happens?

2. Change $Price\{i \in I\}$ to $Price\{i \in J\}$ in the price declaration. What happens?

3. Which step of the production (production of the components, mounting or testing) is limiting the production? Is there a constraint that could be removed without changing the final result?

4. The company optimized its testing procedure and can reduce the testing time by 10% for each of the robot type. How does this affect the final result and why?
5. A market study shows that it is very unlikely to sell more than 50, 30, 40, 20, 15, 10, 60, 20, 17 and 10 units of the 10 robot types. Add the corresponding constraint and explain why the result changes dramatically.

6. Instead of maximizing the total revenue only, we take the material costs into account too. The material costs per type and unit are 120, 120, 20, 15, 9, 30, 10, 27, 270 and 20 respectively. Change the model in order to maximize the revenue.

7. Which type of robot generates the largest revenue? Use an indexed operator to calculate the right answer directly in LPL.

8. Assume that a fourth production step is needed. What do you need to change in the model and where do you see difficulties?

Answer (Question see 3)

1. Running the model, the result is the same as before. To really get 11 products one also needs to extend the data tables.

2. An error occurs, because J is not declared. Note, LPL is case-sensitive, hence, j and J are two different names.

3. Add the three parameters after the minimizing as follows and a Writep statement:

   ```lpl
   parameter C:=\sum_{i \in I} HC[i] * x[i];
   parameter M:=\sum_{i \in I} HM[i] * x[i];
   parameter T:=\sum_{i \in I} HT[i] * x[i];
   Writep(C,M,T);
   ```

   This calculates the used capacities. One can derive from these numbers that the Component and Testing process are at there maximal capacity limit, whereas the Mounting process has a small quantity of capacity left. That means: if we increase the capacity of Mounting, we can not produce more, since the limiting factors are Component and Testing, not the Mounting.

4. The third constraint can easily be replaced by:

   ```lpl
   Testing: \sum_{i \in I} HT[i]*0.9*x[i] <= 3000;
   ```

   Running the model and checking the the parameters of the previous question shows, that the testing capacity is no longer a limiting factor. This allows the company to increase the revenue significantly.

5. We introduce a new parameter and add a new constraint as follows:

   ```lpl
   parameter Demand{i \in I} := [50 30 40 20 15 10 60 20 17 10];
   constraint Dem{i \in I}: x[i] <= Demand[i];
   ```

   It shows that the best strategy is to produce exactly the demanded amount. All the other constraints are not limiting the production anymore and the final revenue is much lower than before.

6. A new parameter is introduced and the objective function has to be adapted:

   ```lpl
   parameter Cost{i \in I} := [120 120 20 15 9 30 10 27 270 20];
   maximize revenue: \sum_{i \in I} (Price[i]-Cost[i]) * x[i];
   ```
7. The indexed operator used to answer this question is: $\text{argmax}\{i \text{ in } I\} \ A[i]$
where $A$ is a new parameter defined as: $\text{parameter } A(i) := \text{Price} \times x$; The argmax operator returns $i$ if the value of the $i$-th element is the largest value in the list. We find that the 9th robot generates the largest revenue.

8. We have to add a new parameter list containing the durations of the fourth step for each model. Furthermore we have to add an additional constraint. With only four steps, this is not a big deal. However, when having a lot of production steps, there exists a much better solution, that will be presented in model tutor05\(^1\).
4 Using Indices II (tutor04)

Problem: This is the same model as tutor03\textsuperscript{2} with some minor syntax differences (simplifications).

Listing 4: The Complete Model in LPL [1]

```lpl
model TUTOR04 "Using Indices II"
set i:=[Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7 Robot8 Robot9 Robot10];
parameter HC(i):=[5 5 4 5 6 5 7 8 4 7];
HM(i) := [4 8 5 6 4 8 7 6 5 3];
HT(i) := [6 2 4 6 3 4 5 2 5 3];
Ordered(i) := [20 15 7 6 5 8 9 8 7 5];
Price(i) := [300 200 100 50 50 100 200 100 400 200];
integer variable Robots{i};
constraint
  Component: sum{i} HC*Robots <= 3500;
  Mounting: sum{i} HM*Robots <= 4800;
  Testing: sum{i} HT*Robots <= 3000;
  Order{i}: Robots >= Ordered;
maximize revenue: sum{i} Price*Robots;
Writep(revenue);
Write(' Robots Rob-Ord Price Robots Tot.Hours\n');
Write{i}(’ %-7s %3d %4d %5d %2d\n’,
  i,Robots-Ordered,Price,Robots,HC+HM+HT);
end
```

LPL Modeling Steps

There are several differences from the previous model tutor03.lpl:

1. The elements of set \( i \) do not need to be integers. The user can give them names such as Robot1 ... Robot10.

2. If the context allows it, there is no need to make a difference between index names and set names. In this model, we use \( i \) for a set name and an index name. the syntax \( i \) in I can be simplified just to \( i \).

3. The indices in expressions, such as \( \text{Robots}[i] \) can be dropped, since LPL already knows that \( \text{Robots} \), for example, has been defined over the index-set \( i \). Hence the two following lines are equivalent for LPL:

   \[ \sum_\{i\} HC \times \text{Robots} \leq 3500 \]
   \[ \sum_\{i\} HC[i] \times \text{Robots}[i] \leq 3500 \]

4. A Write statement may be used to write formatted output. In the model, we use two Write. The first is to output just a string to the NOM-file. The second is to output a formatted line for each \( i \) (explained in a later model). The syntax is similar to C or Java’s printf() function. So, \( %-7s \) means to fill the first parameter \( (i) \) with 7 chars (left aligned), \( %3d \) means to fill in an integer (the expression \( \text{Robots}-\text{Ordered} \)) with 3 positions, and so on.

\textsuperscript{2} http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor03
Question  (Answer see 4)

1. Modify \%3d in the Write just below Robots to \%6.2f and run the model again. What’s the difference?

2. How much is the revenue, if no robot has been ordered in advance?

Answer  (Question see 4)

1. The number of Robots are written with 2 decimal places.

2. 252000. One just need to outcomment the bounds Order(i). The data for Ordered are still in the model but are no used except for the output.
5 Using Indices III (tutor05)

Problem: A second index-set \( j \) for a set of production steps (processes) is introduced. In contrast to the previous model tutor04.lpl, where the production of the robots passed three processing steps (Component, Mounting, Testing), in this model it is supposed that the production goes through 8 steps. Instead of introducing 8 constraints, a second index-set \( j \) is introduced to collect the list of processing steps. Two data change now: a vector Capacity\( \{j\} \) is introduced in place of the right hand side of the constraint, and the three vectors HC\( \{i\} \), HM\( \{i\} \), and HT\( \{i\} \) in the previous model are now replaced by a two-dimensional matrix Hours\( \{i,j\} \). This matrix gives the needed hours for passing the processing step \( j \) for robot type \( i \).

Listing 5: The Complete Model in LPL [1]

```
model TUTOR05 "Using Indices III";
set i := [Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7
         Robot8 Robot9 Robot10];
parameter Ordered\(i\):=[20 15 7 6 5 8 9 8 7 5];
Price\(i\):=[300 200 100 50 50 100 200 100 400 200];
set j:= [Step1 Step2 Step3 Step4 Step5 Step6 Step7 Step8];
parameter Capacity\(j\):=[3500 4800 3000 3400 3000 3200 4000 2500 ];
Hours\(i,j\) := [9 9 9 9 9 9 .9
                   5 8 2 . 3 . . 3
                   4 5 4 . 5 6 . 5
                   5 6 6 . 8 . . 4
                   10 4 3 . . 5 . 6
                   5 8 4 . . . 5 1
                   7 7 5 5 . 3 . . 2
                   8 6 2 4 . . . 5
                   4 5 5 6 4 . . .
                   7 3 3 1 1 1 1 4];
variable Robots\(i\);
constraint
    Steps\(j\): sum\(i\) Hours\(i,j\) * Robots\(i\) <= Capacity\(j\);
    Order\(i\): Robots\(i\) >= Ordered\(i\);
maximize revenue: sum\(i\) Price\(i\) * Robots\(i\);
Writep(revenue, Robots);
end
```

LPL Modeling Steps

The data are now organized differently: All of them are collected in "tables".

1. A second index-set \( (j) \) is introduced for the production steps. The previous models use three explicit production steps, resulting in three constraints. In this model, a generic number of steps - concretely 8 - is used.

2. The time, which indicates the hours required for each type of robot \( i \) at each production step \( j \), is defined as a two-dimensional table Hours\( \{i,j\} \). Undefined (or zero) entries are entered as a dot.

3. The Hours\( \{i,j\} \) matrix is defined as a list of \(|i| \cdot |j| \) entries.
4. Note that the \texttt{Steps\{i\}} constraint generates 8 different constraints. One can verify this when generating the \texttt{EQU-file} (in \texttt{lplw.exe} choose \texttt{Tools/Options enter \textquoteleft e\textquoteright, run, then open menu \textquoteleft Tools/EQU-file\textquoteright}).

\textbf{Question} \hspace{1em} (Answer see 5)

1. Choose \texttt{Tools/Options}, at the tab "Compiler" enter \textquoteleft e\textquoteright, Run the model, then click the menu \texttt{Tools/Create EQU-file}. A listing of all equations opens.

2. For which constraints the capacity is used at 100\%?

3. What big advantage do you see in using one data table instead of using a separate list for every constraint?

4. The production manager wants to produce the same quantity of every type of robot. What do you have to change?

\textbf{Answer} \hspace{1em} (Question see 5)

1. An equation listing is generated. It is stored in the \texttt{EQU-file}. On disk the file is: \texttt{tutor06.equ}.

2. (1) Run the model, (2) click the tab \textquoteleft TABLE\textquoteright, (3) click the blue small cycle C called \textquoteleft Steps\textquoteright at the left on the \textquoteleft Tree\textquoteright tab part of the application, (4) open menu \texttt{Tools/Options}, (5) choose tab \textquoteleft Format\textquoteright, (6)in the \textquoteleft Data Attribute\textquoteright list choose the radio button \textquoteleft Dual / Reduced cost\textquoteright. A reduced table appears with processing steps \texttt{Step1} and \texttt{Step3}. These two constraints are at their maximal capacities. (To understand this you need to know what dual values are.)

3. When you have to add an additional production step or an additional type of robot the above formulation is adapted much faster. Instead of adding a parameter list and an additional constraint you just have to adjust the corresponding set formulation and enter the data into the table. The additional constraint is generated automatically by LPL. With regard to reusability and maintainability, you should always prefer the structure demonstrated in this example.

4. We add a variable \texttt{quantity} and the following constraint:

```
\text{constraint \texttt{Quantity\{i\}: Robots\{i\} = quantity;}}
```

The maximum revenue now is 92968.75. Note that in this example both the variable and the constraint have the same name. However, this is only possible because LPL makes a difference between lower- and uppercase letters.
6 Data Include Files (tutor06)

**Problem:** This is the same model as tutor05.lpl. The data are stored in another file (tutor.inc) and are included at parse-time. The instruction to physically read and include another file into the code is:

```plaintext
| / *$I 'filename' * /
```

The included data file – also specified in LPL syntax – is as follows (note that in the Web browser the include file already directly included in the code):

```plaintext
-- tutor.inc: data file for the tutor models as include-file

set i := [Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7 Robot8 Robot9 Robot10];
    j := [Step1 Step2 Step3 Step4 Step5 Step6 Step7 Step8];
parameter Ordered[i] := [20 15 7 6 5 8 9 8 7 5];
    Price[i] := [300 200 100 50 50 100 200 100 400 200];
    Capacity[j] := [3500 4800 3000 3400 3000 3200 4000 2500 ];
    Hours[i,j] := [
        9 9 9 9 9 9 . 9
        5 8 2 . 3 . . 3 3
        4 5 4 . 5 6 . 5
        5 6 6 . 8 . . 4
        10 4 3 . . 5 . 6
        5 8 4 . . . 5 1
        7 7 5 5 . 3 . 2
        8 6 2 4 . . . 5
        4 5 5 6 4 . . .
        7 3 3 1 1 1 1 4];
```

**Listing 6: The Complete Model in LPL [1]**

```plaintext
model TUTOR06 "Data Include Files";
    /*$I 'tutor.inc' */ — a data file is included here
    variable Robots(i);
    constraint
        Steps[j]: sum(i) Hours*Robots <= Capacity;
        Order[i]: Robots >= Ordered;
    maximize revenue: sum(i) Price*Robots;
    Write('Revenue: %7.3f\n', revenue);
    Write('Variable Lower Bounds:
%10.2f\n', {i} GetValue(Robots,1));
    Write('Variable Upper Bounds:
%10.2f\n', {i} GetValue(Robots,2));
    Write('Dual Values:
%10.2f\n', {i} GetValue(Robots,3));
    Write('Element Names of set i:\n%10s\n', {i}i);
    Write('nVariableNames Values\n----------------------\n');
    Write(i){'-%15s = %6.2f\n', GetName(Robots,0), GetValue(Robots,0));
end
```

21
LPL Modeling Steps

1. At any point, a file can be included using the $i$ option. Inclusion of up to level of five is possible.

2. The second, third and fourth Write statement prints the lower bound, the upper bound, and the dual values of the Robots variable. The function in LPL is defined as GetValue(reference, what) (see manual).

3. The last Write statement prints the variable names with the values: The function GetName() returns the variable names as string.

Question  (Answer see 6)

1. List the right hand side of constraint Steps. List also the reduced costs of Steps.

2. Click the 'Files' tab in lplw in the left part then click tutor.inc.

Answer  (Question see 6)

1. You need to add the instruction

   ```
   Write('RHS Values:
   %10.2f\n', {j} (GetValue(Steps,2));
   Write('RC Values:
   %10.2f\n', {j} (GetValue(Steps,7));
   ```

2. The file tutor.inc opens in a new tabbed window and the content can be edited.
7 Reading text files (tutor07)

**Problem:** The data are read from a text-file `tutor.txt` using the function `Read`.

**Listing 7:** The Complete Model in LPL [1]

```lpl
model TUTOR07 "Reading text files";
set i; j;
parameter Ordered{i}; Price{i}; Capacity{j};
parameter Hours{i,j};
variable Robots{i};
constraint
  Steps{j}: sum{i} Hours*Robots <= Capacity;
  OrderX{i}: Robots >= Ordered;
parameter count;
Read('tutor.txt, %1:Table');
Read{i}('%1', i=1,Price,Ordered);
Read{j}('%2', j,Capacity);
Read{i}('%3', i, {j} Hours, count:=count+1);
maximize revenue: sum{i} Price*Robots;
Writep(revenue, Robots,count);
end
```

The text data file read by LPL has the following structure:

```
// tutor.txt: data file as plain text for the tutor models
// Data for the tutor examples

Table 1: Robots: i, Price, Ordered
    Robot1  300  20
    Robot2  200  15
    Robot3  100  7
    Robot4  50   6
    Robot5  50   5
    Robot6  100  8
    Robot7  200  9
    Robot8  100  8
    Robot9  400  7
    Robot10 200  5

Table 2: Steps: j, and Capacity
    Step1  3500
    Step2  4800
    Step3  3000
    Step4  3400
    Step5  3000
    Step6  3200
    Step7  4000
    Step8  2500

Table 3: Hours
    /* Step1 Step2 Step3 Step4 Step5 Step6 Step7 Step8 */
```
LPL Modeling Steps

The data are separated from the model file and are organized in a text file tutor.txt. This file is organized sequentially in three blocks – also called “Tables” each beginning with the user defined block delimiter string Table. Each Read instruction can read a block. Which block to read is specified by a block number (‘%1’). These blocks are numbered beginning with 1. Each Read opens the file, reads a block and closes the file automatically. There is no need to open and close files in LPL.

1. LPL can read these blocks separately with the Read function. The data within the blocks must be organized in rows and columns. Such files are easy to generate by other applications.

2. The first Read instruction only defines two file parameters: the filename (tutor.txt) and the block delimiter string (Table). These parameters are stored for subsequent Read instructions.

3. The second Read instruction opens the file, then jumps to the first (‘%1’) occurrence of Table that starts a new line and begins reading from the next line. Since Read is indexed over \{i\}, it reads lines until another block delimiter string appears at the beginning of a line or until the end of file has been reached. Each line contains three data separated by blanks. They are recognized and assigned to the internal entities i, Ordered, and Price.

4. A single Read always reads a single line. An indexed Read repeats reading lines from a block. Hence, Read\{i\} repeats the reading. The read elements are separated by commas.

5. If an element to read is indexed then the elements are repeatedly read on the same line. Hence, the code i, \{j\}Hours reads first the i on the line and then repeats reading the Hours[i, j] on the i-th line ordered by j. It is a powerful mechanism to read entire matrices in a single instruction.

6. Furthermore, the last instruction displays an instruction count:=count+1 that is executed after each reading an line from the file.

Question  (Answer see 7)
1. Exchange the second and the third Read instruction. What happens?

2. Comment the maximize instruction then execute. What happens?

**Answer** (Question see 7)

1. Nothing! The reads can be done in any order. There is no need to read the text blocks in sequential order from a text file.

2. The model is not solved. However, that data are nevertheless read from the textfile. This can be seen when adding an instruction Writep(Hours), for example. Another way to see the data is to click the yellow cycle 'M' above 'Tree' in the left part of the lplw window. Clicking on it enlarges the list by green diamond signs (for example). Click on 'Hours' then on the 'TABLE' tab in the menu and the data of Hours are displayed in a grid.
8 Reading data from Excel I (tutor07a)

Problem: The data are read from an Excel sheet (Sheet2) by the instruction Read. Note that lpl should correspond to the Excel Version 32/64-bit to run correctly.

Listing 8: The Complete Model in LPL [1]

```lpl
model TUTOR07a "Reading data from Excel I";
set i; j;
parameter Ordered{i}; Price{i}; Capacity{j};
parameter Hours{i,j};
variable Robots{i};
constraint Steps{j}: sum{i} Hours*Robots <= Capacity;
   OrderX{i}: Robots >= Ordered;
string parameter WB:='rdb:tutor.xls';
Read{i}(WB&',[Sheet2$A5:C15]', i='Robots', Price='Price', Ordered='Ordered');
Read{j}(WB&',[Sheet2$E5:F13]', j='Steps', Capacity='Capacity');
Read{i,j}(WB&',[Sheet2$H5:J62]', i='i', j='j', Hours='Hours');
maximize revenue: sum{i} Price*Robots;
Writep(revenue, Robots);
end
```

Note that the data in the Excel sheet Sheet2 are organized in a similar way than the data in a database (in contrast to the sheet Sheet1). The Excel sheet Sheet2 contains three "tables" ordered in ranges in Excel with a header in each range. Therefore, the data can be read from Excel in the same way as they are read from a regular relational database using a database driver. The whole information of which driver to use is packed by the filename define in WB

```lpl
| string parameter WB:='rdb:tutor.xls';
```

The starting part rdb: says to use a database driver and the file extension says to use the Microsoft Jet driver. The parameter is expanded by LPL to the following connection string (and one could alternately use this connection string):

```lpl
| string parameter WB:='Provider=Microsoft.Jet.OLEDB.4.0;ExtendedProperties="Excel 8.0;HDR=Yes:";Data Source=tutor.xls';
```

(Note that the \ character at the end of the first line as well as all leading blanks are removed by the LPL parser.)

The Excel sheet is shown in Figure 2.

LPL Modeling Steps

The data are separated from the model file and can be read by instruction. The Read instructions are similar than they are in the previous version where data are read from text files. There are differences:

1. An string parameter WB defines the connection string\(^3\) to Excel. This is concatenated with the source filename (here an Excel workbook) do give the string WB. This defines the source of reading. (Note that a string can be broken over lines in LPL using a \ character at the end of a line.)

\(^3\)The concept of connection string is explained at wikipedia or a similar source. The user should be familiar with this concept in order to use database connectivity with LPL.
2. The string `WB` is concatenated with the “table name”, that is, the range within the Excel sheet. A range in Excel is specified by the worksheet name followed by a top-left and bottom-right cell name as follows: `[Sheet2$A5:C15]`. The de filename and table name are separated by a comma.

3. The three following parameters have the form `lpl_id = source_field`. Hence, the expression `i='Robots'` means that the set `i` is assigned from the field (column) with the header name ‘Robots’, this is a standard way to name columns in Excel that act as database tables.

4. It is important to notice that the read elements must be organized into columns or fields. In this case, the columns are named as ‘Robots’, ‘Ordered’, and ‘Price’. For LPL, they also could be accessed through numbers 1, 2, and 3, indicating the column numbering beginning with number 1. Hence, the instruction could also be written as follows:

   ```plaintext
   Read(i) (WB&’,[Sheet2$A5:C15]’, i=1, Price=2, Ordered=3);
   ```
indicating that \( i \) is read from “column” 1 in the indicated sheet range. If the data are *not* organized this way, one can read from Excel using a more direct driver (see model tutor7a1\(^4\)).

5. Since the three elements are listed in increasing numbers beginning with 1, one also can leave them out, LPL just assign them anyway. Hence the instruction can be simplified to:

\[
\text{Read}(i)(\text{WB}‘, [\text{Sheet2}\$A5:C15]‘, i, \text{Price, Ordered});
\]

Now we are basically on the same instruction as for the text reading in the previous model tutor07\(^5\), except that the source name is different.

**Question** (Answer see 8)

1. Replace the first \textit{Read} instruction to:

\[
\text{Read}(i)(\text{WB}‘, [\text{Sheet2}\$A5:C15]‘, i, \text{Price, Ordered});
\]

2. Replace the first \textit{Read} instruction to:

\[
\text{Read}(i)('tutor.txt,%1:Table‘, i, \text{Price, Ordered});
\]

Compare the two statements and compare the source file too.

**Answer** (Question see 8)

1. This verifies that the two instruction are indeed the same.

2. The same data are read from the text file and the Excel sheet. This shows: text file reading and Excel reading are – from the syntactical point of view – basically the same. Only the source parameter has changed. One also can write for text files:

\[
\text{Read}(i)('tutor.txt,%1:Table‘, i=1, \text{Price}=2, \text{Ordered}=3);
\]

This confirms that in text files “tables” are also seen as columns and rows. In text files the columns are delimited by certain “blank” characters. By default, they are *space*, *tabulator* and *linefeed*. However the user can define its own delimiting characters. Common are *comma* or *semicolon*.

The Excel Worksheet can be downloaded here: tutor.xls\(^6\).

\(^4\) http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor7a1
\(^5\) http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor07
\(^6\) http://lpl.virtual-optima.com/lpl/lplmodels/tutor.xls
9 Reading data from Excel II (tutor07a1)

Problem: The data are read from an Excel sheet by the instruction Read. Note that lpl should correspond to the Excel Version 32/64-bit to run correctly.

Listing 9: The Complete Model in LPL

```
model TUTOR07a1 "Reading data from Excel II";
  set  i; j;
  parameter Ordered(i); Price(i); Capacity(j);
  parameter Hours(i,j);
  variable Robots(i);
  constraint
    Steps(j): sum(i) Hours * Robots <= Capacity;
    OrderX(i): Robots >= Ordered;
  Read('tutor.xls,[Sheet1$C3:L3]', {i}i);
  Read('tutor.xls,[Sheet1$C14:L14]', {i}Ordered);
  Read('tutor.xls,[Sheet1$C15:L15]', {i}Price);
  Read(j)('tutor.xls,[Sheet1$A5:A12]', [], j);
  Read(j)('tutor.xls,[Sheet1$N5:N12]', Capacity);
  Read(j)('tutor.xls,[Sheet1$C5:N12]', {i}Hours);
  maximize revenue: sum(i) Price*Robots;
  Writep(revenue, Robots);
end
```

Compared to the data in the previous Excel tutor.xls$Sheet2 (see model tutor07a), the data in this worksheet are organized in a more “free style” (Figure 3. The data are loosely scattered over the sheet.

![Data in Excel Sheet ("free style")](image)

The Excel sheet Sheet1 contains data distributed over several ranges. For example the list of robot types is displayed in the cells C3:L3. The set of production steps is in the range A5:A12, and so on. These ranges can be read by LPL, each with a separate Read statement.

The Excel Worksheet can be downloaded here: tutor.xls.

---

7 http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor07a
8 http://lpl.virtual-optima.com/lpl/lplmodels/tutor.xls
LPL Modeling Steps

The data are separated from the model file and can be read by instruction. The Read instructions are similar to those in the previous version where data are read from text files. There are differences:

1. Each Read instruction contains two parameters: the first parameter defines the Excel Filename and the “table” separated by a comma. The “table” consists of a worksheet name and a range, enclosed within brackets. The second parameter is LPL entity that must be read.

2. Again an indexed Read reads over several “lines” (that is, rows) in a sheet, and an indexed parameter reads cells over several columns.

3. Hence, the parameter {i}Ordered means to read cells on a single row, defined by the range C14:L14 (see second Read).

4. Whereas, Read{j}('tutor.xls,[Sheet1$A5:A12]', j); means to read a single data j on several rows. The logic is similar for other Read statements.

Question  (Answer see 9)

1. What is the main difference between the “free style” and the “database style” of data in Excel?

Answer  (Question see 9)

1. In the “free style”, the data are scattered over the sheet and LPL must read each range separately. In “database style” the tables are organized in a way similar to a database table (with a header as fieldnames) and LPL reads them using a database driver, like ODBC or MS’s Jet engine.
10 Reading data from Database (tutor07b)

**Problem:** The data are read in from an Access database by the instruction `Read`. Note that `lpl` should correspond to the MS Access Version 32/64-bit to run correctly.

Listing 10: The Complete Model in LPL [1]

```lpl
model TUTOR07b "Reading data from Database";
  set i; j;
  parameter Ordered{i}; Price{i}; Capacity{j};
  parameter Hours{i,j};
  variable Robots{i};
  constraint
    Steps{j}: sum{i} Hours * Robots <= Capacity;
    OrderX{i}: Robots >= Ordered;
  string parameter DB:='Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=tutor.mdb';
  Read{i}(DB&',iTable', i='Robots', Price='Price', Ordered='Ordered');
  Read{j}(DB&',jTable', j='Steps', Capacity='Capacity');
  Read{i,j}(DB&',HoursTable', i='i', j='j', Hours='Hours');
  maximize revenue: sum{i} Price*Robots;
  Writep(revenue, Robots);
end
```

The MS Access database `tutor.mdb` consists of three tables as shown in Figure 4. It can be downloaded here: `tutor.mdb`⁹.

LPL Modeling Steps

The data are separated from the model file and can be read by instruction. The `Read` instructions are similar to those in the previous versions where data are read from text files and Excel sheets. There are differences:

1. An **string parameter** `DB` defines the **connection string** to a database. ¹⁰ In LPL, the connection string is derived from the file extension. In this case it is `.mdb` that is a Access database and so the parameter is automatically expanded to:

   ```lpl
   string parameter DB:='Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=tutor.mdb';
   ```

   This is concatenated with the source filename (here an database filename) do give the string `DB`. This defines the source of reading. (Note if the file extension would have been `.db` then LPL would read from a SQLite database, see reference manual for more details.

2. The string `DB` is concatenated with a database **table name** or a SQL SELECT statement that delivers a table. This is `iTable`, for example, for the first `Read` statement which contains the three fields `Robots`, `Price`, and `Ordered`. The database filename and table name are separated by a comma. Database filename and table name together specify the source of reading databases for LPL.

---


¹⁰ The concept of **connection string** is explained at [wikipedia](https://en.wikipedia.org/wiki/Database_connection_string) or a similar source. The user should be familiar with this concept in order to use database connectivity with LPL.
3. The three following parameters have the form lpl_id = source_field. Hence, the expression i='Robots' means that the set i is assigned from the field with the name 'Robots' in the database table.

4. It is important to notice that the read elements are organized into fields with a certain ordering. In this case, the fields are named as 'Robots', 'Ordered', and 'Price' in this order. For LPL, they also could be accessed through numbers 1, 2, and 3, indicating the column numbering beginning with number 1. Hence, the instruction could also be written as follows:

   Read{i}(DB&',iTable', i=1, Price=3, Ordered=2);

indicating that i is read from column 1 - (Note that Price is the third fields no the second!)

5. Reading the three elements in increasing numbers beginning with 1, one also can leave them out, LPL just assigns them anyway. Hence the instruction can be simplified to:

   Read{i}(DB&',iTable', i, Ordered, Price);

(Note that Ordered is before Price in the database table.) Now we have basically the same instruction as for the text reading in the previous model tutor07. Only the source name is different.

---

11 http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor07
**Further Comments:** Sometimes it makes sense to read several fields into the same matrix within LPL. Suppose the database above contains an additional table which is depicted in Figure 5.

![Image of Table HoursTable1 in the database](image)

**Figure 5:** Table HoursTable1 in the database

This table `hoursTable1` contains the same information as the table `hoursTable`, but the layout is different. The production steps `j` are defined as fields `Step1` to `Step8`. We would like – similar to the third table in model `tutor07` – to read all these fields into a single matrix parameter `H(i, j)` (which is identical to the table `Hours(i, j)` in our model). This can be achieved by the following reading statement:

```
Read(i)(DB&', 'HoursTable1', i='I', {j} H);
```

Note that the syntax is similar to the syntax that is used in the model `tutor07` for reading the `Hours` table. The condition for this to work is, of course, that the fields are in the same order as the ordering of the elements in `j`.

**Question** (Answer see 10)

1. Replace the first `Read` instruction to:
   ```
   Read(i)(DB&', iTable', i, Ordered, Price);
   ```

2. Replace the first `Read` instruction to:
   ```
   Read(i)('tutor.txt,%1:Table', i, Price, Ordered);
   ```

   Compare the two statements and compare the source file too.

3. The table name can also be a SQL `SELECT` statement. Modify the LPL `READ` instruction in that sense.

**Answer** (Question see 10)

1. This verifies that the two instructions are indeed the same.

2. The same data are read from the text file and the Excel sheet. This shows: text file reading and Excel reading are basically the same. Only the source parameter has changed. One also can write for text files:

   Read(i)(‘tutor.txt,%1:Table’, i=1, Price=2, Ordered=3);

   This confirms that in text files “tables” are also seen as columns. In text files the columns are delimited by certain “blank” characters. By default, they are space, tabulator and linefeed. However the user can define its own delimiting characters. Common are comma or semicolon.

3. The instruction is:

   Read(i)(DB&’,SELECT Robots,Price,Ordered FROM iTable’, i, Price, Ordered);

   (Note that the ordering of the fields Price and Ordered are reversed.)
11 Writing to text files (output locally only) (tutor08)

**Problem:** Data and the solution are written to a text file using the function `Write`.

Listing 11: The Complete Model in LPL [1]

```lpl
model TUTOR08 "Writing to text files (output locally only)");
/*$I 'tutor.inc' */ -- a data file is included here
variable Robots{i};
constraint
  Steps{j}: sum{i} Hours*Robots <= Capacity;
  OrderX{i}: Robots >= Ordered;
maximize revenue: sum{i} Price*Robots;
Write('tutor08out.txt','--LPL generated file

Table 1 -- robot types
  Robot1 300 20
  Robot2 200 15
  Robot3 100 7
  Robot4 50 6
  Robot5 50 5
  Robot6 100 8
  Robot7 200 9
  Robot8 100 8
  Robot9 400 7
  Robot10 200 5

Table 2 -- processing steps
  Step1 3500
  Step2 4800
  Step3 3000
  Step4 3400
  Step5 3000
  Step6 3200
  Step7 4000
  Step8 2500

Table 3 -- unit time consumption

Table 4 -- Solution

end
```

The text output file written by LPL has the following structure:

---LPL generated file

**Table 1 -- robot types**
- Robot1 300 20
- Robot2 200 15
- Robot3 100 7
- Robot4 50 6
- Robot5 50 5
- Robot6 100 8
- Robot7 200 9
- Robot8 100 8
- Robot9 400 7
- Robot10 200 5

**Table 2 -- processing steps**
- Step1 3500
- Step2 4800
- Step3 3000
- Step4 3400
- Step5 3000
- Step6 3200
- Step7 4000
- Step8 2500

35
Table 3 -- unit time consumption

<table>
<thead>
<tr>
<th>Robot</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>0</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot2</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Robot3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Robot4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Robot5</td>
<td>10</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Robot6</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Robot7</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Robot8</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Robot9</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Robot10</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4 -- Solution

<table>
<thead>
<tr>
<th>Robot</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot1</td>
<td>20</td>
</tr>
<tr>
<td>Robot2</td>
<td>266</td>
</tr>
<tr>
<td>Robot3</td>
<td>7</td>
</tr>
<tr>
<td>Robot4</td>
<td>6</td>
</tr>
<tr>
<td>Robot5</td>
<td>5</td>
</tr>
<tr>
<td>Robot6</td>
<td>8</td>
</tr>
<tr>
<td>Robot7</td>
<td>9</td>
</tr>
<tr>
<td>Robot8</td>
<td>8</td>
</tr>
<tr>
<td>Robot9</td>
<td>420</td>
</tr>
<tr>
<td>Robot10</td>
<td>5</td>
</tr>
</tbody>
</table>

LPL Modeling Steps

The instruction `Write` can be used to write formatted text files.

1. The parameters of the function `Write` are as follows: The first parameter is a filename. The second parameter is a format string, and the other parameters are the data to write. The syntax is: `Write(filename, format, datalist)`.

2. Each `Write` instruction opens the file automatically and closes it at the end of the writing.

3. If the first parameter `filename` is missing then the file is opened in an appending mode, that is, the data are added automatically at the end of the existing file which name has been defined by a previous `Write` statement.

4. If no previous `Write` has defined a filename then the file name is the NOM-file.

5. If the first parameter `filename` is *not* missing then the file is opened in a write mode, that is, an existing file with the same name is erased before writing can begin.

6. The first parameter is missing if and only if the `Write` has only one parameter or if the first (string) parameter contains a linefeed or a `%` character.

7. The second parameter `format` specifies the format of the output. It’s a string. This parameter of the first `Write`, for example, is:
It says to write the text --LPL generated file to the file, then to write two linefeed characters (\n), then to write Table 1 and another linefeed.

8. The format parameter of the second Write is: %9s\t%4d\t%2d\n
It says to translate i into a string of length 9 ( %9s), then to write a tabulator \t, then to replace %4d with Price with length 4, write another tab, and finally, to write Ordered with length 2 ( %2d).

Note: The model tutor08e lists all format specifiers that can be used in LPL.

Question (Answer see 11)

1. Replace the second Write statement with:

   \[
   \begin{array}{l}
   \text{Write\{i\}(\%-9s\t%6.2f\t%2d\n\text{, i, Price, Ordered});}
   \end{array}
   \]

2. Study all specifiers!

3. Why do all Write statements send the data to a single file named tutor08out.txt?

Answer (Question see 11)

1. The set i is written in left align mode. Price is written as floating point number with two decimals.

2. Open the model tutor20.

3. Only the first Write specifies the filename. This filename is stored for subsequent Write instructions. Since the first parameter of all other Write instructions contain the character \n, they are all interpreted by LPL as the format parameter and data is appended to the file.
12 Writing data to Excel (tutor08a1)

**Problem:** The data are written to an Excel sheet by the instruction `Write`.

Listing 12: The Complete Model in LPL [1]

```lpl
model TUTOR08a1 "Writing data to Excel";

/*$I 'tutor.inc' */ — a data file is included here

variable Robots{i};

constraint
  Steps{j}: sum{i} Hours * Robots <= Capacity;
  OrderX{i}: Robots >= Ordered;

Write('tutor08out.xls,[Sheet4$C3:L3]', {i}i);
Write('tutor08out.xls,[Sheet4$C14:L14]', {i}Ordered);
Write('tutor08out.xls,[Sheet4$C15:L15]', {i}Price);
Write(j){'tutor08out.xls,[Sheet4$A5:A12]', j};
Write(j){'tutor08out.xls,[Sheet4$N5:N12]', Capacity};
Write(j){'tutor08out.xls,[Sheet4$C5:L12]', {i}Hours};

maximize revenue: sum{i} Price*Robots;
Writep(revenue, Robots);
end
```

The `Write` instructions has exactly the same syntax as the `Read` instruction when reading from Excel (see model tutor07a116): only the keyword `Read` has been replaced by the keyword `Write` (and another file name — tutor08out.xls — is used).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Step1</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>3500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Step2</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>4800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Step3</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Step4</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>3400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Step5</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>3000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Step6</td>
<td>9</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Step7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Step8</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>2500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>15</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>300</td>
<td>200</td>
<td>100</td>
<td>50</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>100</td>
<td>400</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The generated Excel sheet Sheet4 is displayed in Figure ???. Note that if the Excel file does not exist, then a new workbook is created. Furthermore, if the corresponding worksheet (here Sheet 4) within the Excel workbook does not exist, it is added automatically.

**Question** (Answer see 12)

1. Is it possible to write data to Excel in a “database style” in LPL?

**Answer** (Question see 12)

1. Unfortunately, this is not possible, because the MS Jet database driver does not correctly interpret a range such as [Sheet1$A5:C15] as a true database table.

---

13 Writing to databases (output locally only) (tutor08b)

**Problem:** Data and the solution are written to a database using the function `Write`. The `Write` statements in the model create a new database with the structure that is depicted in Figure 7.

Listing 13: The Complete Model in LPL [1]

```lpl
model TUTOR08b "Writing to databases (output locally only)"
/* $I 'tutor.inc' */ — a data file is included here

variable Robots{i};

constraint
    Steps{j}: sum{i} Hours*Robots <= Capacity;
    OrderX{i}: Robots >= Ordered;
maximize revenue: sum{i} Price*Robots;

string parameter DB:="tutor08out.mdb";
Write{i}('&:'&DB&',iTable', 'Robots'=i, 'Price'=Price,'Ordered'=Ordered);
Write{i}('*:'&DB&',rTable', 'Robots'=i, 'Quan'=Robots);
Write{j}('*:'&DB&',jTable', 'Steps'=j, 'Capacity'=Capacity);
Write{i,j}('*:'&DB&',HoursTable', 'i'=i, 'j'=j, 'Hours'=Hours);
Write{i}('*:'&DB&',HoursTable1', 'I'=i, {j} (j=Hours));
end
```

Figure 7: The generated Database

In a way similar to the `Read` statement, LPL connects to a database using the `connection string` method. The parameter `DB` contains that string together with the database name `tutor08b.mdb`.
LPL Modeling Steps

The instruction Write can be used to write to existing database tables or can also create them and fill them on the fly.

1. The parameters of the function Write are as following: The first parameter is a connection string with the filename and contains also a tablename separated by a comma. These parts form the destination and it is a string, and the other parameters are the data to be written. The syntax is:
   \[
   \text{Write}(\text{destination}, \text{datalist}) \quad \text{(there is no format parameter as we have for the text file writing)}.
   \]

2. Each Write instruction opens the database automatically and closes it at the end of the writing.

3. If the first parameter destination begins with ‘&:’ then this means that the database together with the table is directly created (erasing a existing file with the same name). If it begins with ‘*:’ then the existing database is extended with a new table. If it begins with ‘-:’ then this means that the existing table content is deleted before new records are added. And finally, if it begins with ‘+:’, records are added to an existing content.

4. The other parameters assign the data to the specified fields: So ‘Robots’=i means to write the data of the set \( i \) in to the fields named ‘Robots’.

5. Since the first Write statement is indexed over \( i \), this means that LPL write as many records into the table \( i\text{Table} \) as \( i \) has elements. Each writing is an single record writing.

6. Note that the syntax of reading a record from a database using Read is similar (tutor07b\textsuperscript{17}). The big difference is the inverse assignment: instead of \( i=\text{Robots} \) here we write ‘Robots’=i, suggesting that the “traffic” goes from right to left as in an assignment statement.

7. Looking at the last Write statement, one can see that LPL also can create fields from an LPL set. The field names are directly taken from the element name of set in LPL and assigned correspondingly.

Further Comments: There are some other interesting aspects in creating databases through LPL. If the fieldname begins with the character ‘I’, for example, then LPL generates the field as a key field. If it begins with a underscore ‘_’ the field will be indexed automatically in the database.

Question  (Answer see 13)

1. Combine the first two Write statements into a single one.

2. Add a second Write statement as follows:
   \[
   \text{Write}(i)(\text{'+:'&DB&',iTable', 'Robots'=i, 'Price'=Price,'Ordered}
   \]

\textsuperscript{17} \url{http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor07b}
What happens?

3. Replace the first Write statement by the following two statements:

   ```
   Write(i)('&:'&DB&',iTable', 'IRobots'=i, 'Price'=Price,'Ordered '
   '=' Ordered);
   Write(i)('+:'&DB&',iTable', 'IRobots'=i, 'Price'=Price,'Ordered '
   '=' Ordered);
   ```

   What happens?

Answer (Question see 13)

1. The statement is simply:

   ```
   Write(i)('&:'&DB&',iTable', 'Robots'=i, 'Price'=Price, 'Ordered '
   '=' Ordered, 'Quan'=Robots);
   ```

   One only needs to add a field for the solution. There is no need to have two tables.

2. The table iTable is filled two times and contains 20 records, each record twice.

3. As in the previous question, the table iTable will be tried to fill twice. However the database integrity does not allow this since the field ‘IRobots’ is now considered as a key field with unique entries only. The second Write will generate an database SQL exception error and the execution is stopped. The log file of LPL gives more details about the error. Look at it!
14 Creating a Report (report locally only) (tutor08c)

Problem: This model creates a report using the Fast Report library (see FastReport). The LPL model is almost exactly the same as the model tutor08b\textsuperscript{18}. Data and the solution are written to a database using the function Write. The Write statements in the model create a new database with the structure that is depicted in Figure 8.

Listing 14: The Complete Model in LPL [1]

```lpl
model TUTOR08c "Creating a Report (report locally only)";
/*$1 'tutor.inc' */ /* a data file is included here
variable Robots{i};
constraint
  Steps{j}: sum{i} Hours*Robots <= Capacity;
  OrderX{i}: Robots >= Ordered;
maximize revenue: sum{i} Price*Robots;
string parameter DB:='tutor08out.db';
Write('&:'&DB&',tutor08c1,*', 'a'='A Report', 'b'='Production', 'c'='Version 1', 'd'=Now());
Write('*:'&DB&',hTable', 'a'='Data Tables', 'b'='Production of Robots', 'c'='Version 1', 'd'=Now());
Write('*:'&DB&',i0Table', 'Robots'='ID', 'Price'='Price','Ordered'='Ordered', 'Quan'='Quantity');
Write(i)('*:'&DB&',iTable', 'Robots'=i, 'Price'=Price,'Ordered'='Ordered', 'Quan'=Round(Robots));
Write('**:'&DB&',j0Table', 'Steps'='ID', 'Capacity'='Capacity');
Write(j)('*:'&DB&',jTable', 'Steps'=j, 'Capacity'=Capacity);
Write('*:'&DB&',HoursTable0', 'ID'='ID', {j} ('a'&(j+0)=Format('%s ', j)));
---Write{i,j}('*:'&DB&',HoursTable', 'i'=i, 'j'=j, 'Hours'=Hours);
Write(i)('*:'&DB&',tutor08c2,**.pdf', 'I'=i,{j} (j=Hours));
end
```

Figure 8: The generated Database

\[\text{http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor08b}\]

42
In a similar way than the Read statement, LPL connects to a database using the connection string method. The parameter DB contains that string together with the database name tutor08out.db.

**LPL Modeling Steps**

In addition, a report in PDF is generated based on the data written to the database. The crucial modifications are:

1. The first parameter of the first Write statement contains additional comma followed by a star: ‘&:’&DB’,tTable,*’. LPL generates a database table tTable, and in addition, a Fast Report template page (file) called tTable.fr3 that is editable with Fast Reports template designer. In LPLW this template page can be opened by selecting menu View/Open Report Designer and then selecting the file tTable.fr3. One can edit the page at will.

2. If the LPL model is executed a second time, then the template page is not created again, but opened as is.

3. Furthermore, the first parameter of the last Write statement contains also a star and four part the output file of the report.

   ‘*:’&DB’,HoursTable1,*,tutor08out.pdf’

   This statement creates a database table called HoursTable1 and in addition a template page HoursTable1.fr3. This page also can be opened using LPLW menu entry View/Open Report Designer. A preview of the designer page if shown in Figure 9.

4. The four part in the first parameter (which is: tutor08out.pdf) is the filename of the report file.

5. If the extension of the report file were rtf – hence, tutor08out.rtf – then automatically a RTF (MS-Word readable) format file would be generated.

**Question** (Answer see 14)

1. Replace *.pdf by *.rtf, and run the model. What happens?

2. Open the template file tutor08cl.fr3 and modify something. Run again.

**Answer** (Question see 14)

1. A RTF-file is generated that can be opened by MS Word (or LibreOffice Writer).

2. Open the template tutor08cl.fr3 (use lplw.exe menu item <View/Open Report Designer>). In the designer click on the memo tutor08cl_d. On the property page change the DisplayFormat to Date/Time. Save and exit the template. Run the model again. The modified report is generated to file tutor08cl.pdf.
Figure 9: Fast Report Template Page
15 Writing With Formatted Masks (tutor08d)

Problem: This is the same model as tutor07.lpl, except that we use a unit cost per Hours production and the profit is maximized. However, now we use a more complicated masks for model output using the Write statement.

Listing 15: The Complete Model in LPL [1]

```
model TUTOR08d "Writing With Formatted Masks";
  /*$ I 'tutor.inc' */ /* read the data
  parameter CpH := 5 "Cost per hour";
  variable Robots{i};
  constraint
    Steps{j}: sum{i} Hours * Robots <= Capacity;
    Order{i}: Robots >= Ordered;
  maximize profit: sum{i} Price*Robots - sum{i,j} CpH*Hours*Robots;
  Write(' OUTPUT of the TUTOR08d model
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−
    type of number of already Price Cost
    robot robot ordered unit unit
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    n 
    Write{i}(i,'%7s %4d %3d %3d %3d %15s
',
    Robots,Ordered,Price, sum {j}CpH*Hours,
    if (Price<sum {j}CpH*Hours,'loosing money','ok'));
    Write(' Total of revenue : %9.2f
    Total of costs : %9.2f
    Total of profit : %9.2f
    $%
    _
    sum {i}Price*Robots,
    sum {i,j}CpH*Hours*Robots,
    sum {i}Price*Robots - sum {i,j}CpH*Hours*Robots);
end
```

LPL Modeling Steps

1. The Write statement can be used to print formatted text defined by the user distributed over several lines. The mask contains line feed chars (\n) to write a new line. At the end of a mask line we use a back-slash char \ to remove the following line feed and the beginning blank chars on the following line within the mask.

2. The Write expression instructs LPL how to fill this mask.

Note: The model tutor08e lists all format specifiers that can be used in LPL.

Question (Answer see 15)

1. Double the Price vector and see what happens.

Answer (Question see 15)

19 http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/tutor08e
1. To double the prices, one only needs to add an instruction before the declaration of the variables, for example. Hence, add the line

```
|   Price(i) := 2*Price;
```

Then run the model again. The result shows that we are loosing money on Robot 4 and Robot 5 only now.
Problem: This model shows all output formatting options with the Write statement as explained in the reference manual of LPL in Chap. 9.2.2. These options correspond to the specifications which are used by other languages such as Java.

Note that these specifiers can also be used in the Format function that returns the formatted string. An example can be seen at the end of the LPL code.

Listing 16: The Complete Model in LPL [1]

```lpl
model TUTOR08e "Write-Format Examples";
  parameter a := 1234.56;
    integer i := 1234;
    string b := 'abcdefg';
    date d := @2004-11-13T10:30:15;
  Write('d (integer) : %d\n' , -i);
  Write('u (unsigned) : %u\n' , i);
  Write('h (hexadecimal) : %x\n' , i);
  Write('o (octal) : %o\n' , i);
  Write('f (float) : %f\n' , a);
  Write('f (width 5, 1 dec): %5.1f\n' , a);
  Write('f (right adjusted): %-15.3f\n' , a);
  Write('f (left adjusted) : %15.3f\n' , a);
  Write('e (e-notation) : %.2e\n' , a);
  Write('e (e-notation) : %16.7e\n' , a);
  Write('g (same as e or f): %16.7g\n' , a);
  Write('n (with 1000 sep) : %n\n' , a);
  Write('n (4 decimals) : %.4n\n' , a);
  Write('m (currency) : %m\n' , a);
  Write('m (7,3 width) : %7.3m\n' , a);
  Write('b (boolean true) : %b\n' , 1);
  Write('b (boolean false) : %b\n' , 0);
  Write('s (string) : %s\n' , b);
  Write('s (width 10,5) : %10.5s\n' , b);
  Write('z (fraction) : %z\n' , .75);
  Write('z (fraction) : %z\n' , 6.125);
  Write('z (fraction) : %z\n' , 6.875);

  Write('\n--- date now---\n');
  Write('tH (Hour) : %tH\n', d);
  Write('tk (Hour) : %tk\n', d);
  Write('tI (Hour) : %tI\n', d);
  Write('tl (Hour) : %tl\n', d);
  Write('tm (Min) : %tm\n', d);
  Write('tS (Secs) : %tS\n', d);
  Write('tL (mSecs) : %tL\n', d);
  Write('tp (ampm) : %tp\n', d);
  Write('tB (month) : %tB\n', d);
  Write('tb (month) : %tb\n', d);
  Write('th (month) : %th\n', d);
  Write('tm (month) : %tm\n', d);
  Write('tA (wday) : %tA\n', d);
  Write('ta (wday) : %ta\n', d);
  Write('tY (year) : %tY\n', d);
  Write('ty (year) : %ty\n', d);
```

47
Write('td (day) : %td\n', d);
Write('te (day) : %te\n', d);
Write('tR (h:m) : %tR\n', d);
Write('tT (h:m:s) : %tT\n', d);
Write('tr (h:m:s) : %tr\n', d);
Write('tD (y-m-d) : %tD\n', d);
Write('tR (y-m-d) : %tF\n', d);
Write('tc (all) : %tc\n', d);

string parameter mydate:=Format('DAY:(%tc) TIME:(%tT)', d, d);
Write('nThe complete date is: %s\n', mydate);
set s:=1..5;
Write('The complete date is: %s\n', {s} Format('%2s^2=%1d ',s, s^2));

end

The Format function is especially useful in combination within a Write. The last Write in code shows how. The Format function returns a string. This string is concatenated over all elements of s and returned as parameter to the Write, that outputs the whole as a string.
17 Submodels (output locally only) (tutor09)

**Problem:** This model shows how to arrange a model into several submodels that can be called by the main model.

Listing 17: The Complete Model in LPL [1]

```
model TUTOR09 "Submodels (output locally only)";
    mydata;
    writedata;
    ClearData(TUTOR09);
    readdata;
    set i; j;
    parameter a(i); b(j);
    string parameter s[i];

    model mydata "define some data";
        SetRandomSeed(1);
        i := 1..10;
        a[i] := Trunc(Rnd(1,10));
        s[i] := ['AA','BB','CC','DD','EE','FF','GG','HH','II','JJ'];
        j := 1..11;
        b[j] := j^2;
    end

    model writedata "write the data to a file";
        Write('tutor09out.txt','-- Data generated by LPL

        Write('Table 1 -- (i a s)

        Write{i}(' %3s %3d %4s
', i,a,s);
        Write('\nTable 2 -- (j b)

        Write{j}(' %3s %4d
', j,b);
    end

    model readdata;
        Read('tutor09out.txt,%1:Table');
        Read{i}('%1', i,a,s);
        Read{j}('%2', j,b);
    end
end
```

**LPL Modeling Steps**

The model contains three submodels, called mydata, writedata, and readdata. The main model (TUTOR09) calls them in this order. It also calls a function Cleardata(TUTOR09) that clears all the data defined in the main model.

The main model also declares some tables: a set i and j, two numerical vectors a{i} and b{j}, and a string vector s[i].

The three submodel can access and modify the ‘global’ data defined in the main model. The first submodel mydata sets the random seed and fills all the data vectors. Note that a table assignment such as *s(i) := [ ... ] must begin with a star, to specify that the data are assigned through an internal table an not through an expression.

The submodel writedata write the data to a file subdivided into two blocks.

The submodel readdata reads these data again from that file.

The generated file is as follows:

**Question** (Answer see 17)
1. Replace `SetRandomSeed(1)` by `SetRandomSeed(2)`. What happens?

**Answer** (Question see 17)

1. A different data set is now generated for \( a_i \), since this the only vector generated randomly, using the function `Rnd` that returns a uniformly distributed random number.
Sparse Tables (tutor10)

Problem: This model shows sparse data tables based on expressions.

Listing 18: The Complete Model in LPL [1]

```lpl
model TUTOR10 "Sparse Tables";
set i; j;
parameter Ordered{i}; Price{i}; Capacity{j};
    Hours{i, j | i<>10 or j<>4};
variable Robots{i | Ordered>6};
    -- read the data here
Read('tutor.txt,%1:Table');
Read{i}('%1', i,Price,Ordered);
Read{j}('%2', j,Capacity);
Read{i}('%3', i, {j}Hours);
constraint
    Steps{j} : sum{i} Hours * Robots <= Capacity;
    -- Steps{j | j<=3} : sum{i} Hours * Robots <= Capacity;
Order{i} : Robots >= Ordered;
maximize profit : sum{i} Price * Robots;
Writep(profit,Robots,Hours);
end
```

LPL Modeling Steps

Sparsity is one of the key concepts in LPL. Multidimensional tables are stored and handled in a sparse way, which means that only a subset of the Cartesian Product of table elements are stored. A four-dimensional table of only 1000 elements in each dimension goes beyond every memory allocation, if considered as full table. It is, therefore, essential to deal with sparsity. LPL has efficient ways to deal with it.

1. An index-set can be limited by an arbitrary expression, for example. The following declaration

   ```lpl
   variable Robots{i | Ordered > 6};
   ```

   means that a variable Robots is declared for every element in i such that the order is larger then 6. In mathematical notation:

   \[ x_i, \quad \forall \{i| i \in I = \{1 \ldots n\} , O_i > 6\} \]

   In our case it means that the three variables Robot[4], Robot[5] and Robot[10] are discarded from the model.

2. A tuple list \( \{i, j\} \) also represents a set and can, therefore also be limited by expressions. Hence, the declaration:

   ```lpl
   Hours{i,j | i<>10 or j<>4};
   ```

   means that there exists a value within table Hours for each combination of \( (i, j) \in I \times J \) such that \( i \neq 10 \) or \( j \neq 4 \).
3. The condition $i<>10 \text{ or } j<>4$ excludes one tuple: 'declare Hours for all tuples $(i,j)$ except for $i = 10$ and $j = 4$. (The number 10 and 4 indicate the positions of the elements within the sets. Sets are considered always as ordered in LPL.) (The condition could also be written as $\neg(i=10 \text{ and } j=4)$. If the expression evaluates to zero, it is interpreted as 'tuple does not exist', else as 'tuple does exist'.

4. Any index-list may be followed by a condition. Suppose we want the maximizing function only summed up over all Robots yielding a price greater than 100. We could then write the maximizing function as follows:

$$\text{maximize profit: } \sum\{i \mid \text{Price}>100\} \text{ Price } \ast \text{Robots;}$$

**Question**  (Answer see 18)

1. How does the model change, if one modifies the condition $\text{Ordered}>5$ to $\text{Ordered}>6$ in the variable declaration?

2. Change the constraint definition $\text{Steps}\{j\}: \ldots$ to $\text{Steps}\{j|j<=3\}\ldots$. Analyse the solution.

**Answer**  (Question see 18)

1. The variable $\text{Robot4}$ is removed from the model. Note that it is enough to remove it at the declaration. Summation over "all" $i$ then discards them automatically too.

2. The solution did not change! Why? This is because all the constraints $\text{Steps}[4]$ to $\text{Steps}[8]$ are not “tight”, that is, they are not at their maximal capacity.
19 Predefined Functions (tutor11)

**Problem:** In this model, all data are generated by a random generator. The model shows and explains several functions used in LPL.

Listing 19: The Complete Model in LPL [1]

```lpl
model TUTOR11 "Predefined Functions";
  SetRandomSeed(1);
  set i := [1..10]; j := [1..8];
  parameter
    Ordered(i) := Trunc(Rnd(3,20));
    Price(i) := Trunc(Rnd(100,200));
    Capacity(j) := Trunc(Rnd(3000,5000));
    Hours(i,j) := Trunc(Rnd(0,9));
  variable
    Robots(i);
  constraint
    Steps(j): sum(i) Hours * Robots <= Capacity;
    Order(i): Robots >= Ordered;
  maximize profit: sum(i) Max(Price,100) * Robots;
  /* another formulation is:
    profit: sum(i) If(Price < 100,100,Price) * Robots; */
  Writep(profit,Robots);
end
```

LPL Modeling Steps

All data are generated by a random generation procedure.

1. The seed of the random generator is initialized by `SetRandomSeed(1)`.

2. The function `Rnd(a,b)` generates a uniform number between `a` and `b`. The function `Trunc(a)` truncates the number `a` to an integer.

3. Other similar functions are `Ceil(x)` (returns the largest integer smaller than `x`), `Round(x)` (returns the rounded integer of `x`), `Sin(x)` (returns the sinus of `x`), `Abs(x)`, `Tan()`, `Cos()`, `Log()`, and many others.

4. The `Max` function returns the larger of the two operands. Hence, `Max(12,10)` means 12. The smaller of function is `Min`.

5. The `if` function takes at least two arguments. The first is a condition. The second argument is evaluated if the condition evaluates to non-zero (true), else the third argument is evaluated. If the third argument is missing, zero is assumed. Example: the statement `a := if(3=4 , 4 , 3);` will assign 3 to `a`.

6. The `if` can have more than three arguments which is interpreted like a `switch` statement in C. For example:

   ```lpl
   If(a,b,c,d,e,f,g)
   ```

   means:"if `a` is true (non-zero) return `b`, else if `c` is true return `d` else if `e` is true return `f`, else return `g`. Of course, all the parameters `a` to `g` can be arbitrary expressions.
Question  (Answer see 19)

1. How can the prices be generated as a normal distributed vector with mean 100 and a deviation of 20?

2. Sum all prices that are larger than 120 and assign the value to a new parameter.

Answer  (Question see 19)

1. Use the function $\text{Rndn}(m,d)$, it returns a normally distributed random number with mean $m$ and standard deviation $d$. The statement is:

   $\text{Price}(i) := \text{Rndn}(100,20);$ 

2. The statement is as follows:

   $\text{parameter } A := \text{sum}(i|\text{Price}>120) \text{ Price;}$

   There is another way to express the same expression as follows:

   $\text{parameter } A := \text{sum}(i \text{ if}(\text{Price}>120,\text{Price});$

   Note the if with two parameters returns zero as default if the condition is false (equal to zero).
20 Index Operators (tutor12)

**Problem**: Indexed operators are a powerful mean to concisely write large expressions. In mathematical notation, for example, one can write

\[ \sum_{i \in \{1 \ldots 1000\}} x_i \]

which is a shortcut for

\[ x_1 + x_2 + x_3 + \ldots + x_{999} + x_{1000} \]

The operator \( \sum \) is called *indexed operator* for summig up over a set. There exist other such operators, for example max (that returns the largest element in a list) or argmin (which returns the list position of the minimal element). In LPL, these operators can be used in the same way.

Listing 20: The Complete Model in LPL [1]

```lpl
model TUTOR12 "Index Operators";
/∗$I 'tutor.inc' ∗/
set i1{i} :=
sum{j}Hours>24 and (forall{j}(Hours<8) or exist{j}(Hours=0));
variable Robots{i|i1};
constraint
   Steps{j}: sum{i} Hours*Robots <= Capacity;
   Order{i}: Robots >= Ordered;
maximize profit: sum{i} Price*Robots;
parameter a:=max{j} min{i} Hours;
Write(profit,Robots);
Write{i|sum{j}Hours>24}('%s uses more than 24h to be produced\n',i);
Write{i|forall{j}(Hours<8) or exist{j}(Hours=0)}('%s used less then 8h in each step\n',i);
Write{i|exist{j}(Hours=0)}('%s does not use some steps\n',i);
Write{j|atleast(5){i}(Hours>5)}('In %s at least 5 robots use more than 7h\n',j);
end
```

LPL Modeling Steps

In various locations of the model we use index operators.

1. The expression \( \sum{i} Price \cdot Robots \), used in previous models, means the same as \( \sum_i Price_i \cdot Robots_i \).

2. Two other operators are used: forall and exist. Let’s explain the expression containing them.

3. The partial expression \( \sum{j} Hours[i,j] \), calculates the number of hours used to produce a Robot \( i \) (\( \sum_j H_{i,j}, \ \forall i \)). It returns a value for each \( i \).

The expression \( \sum{j} Hours[i,j] > 24 \) returns for each \( i \) true or false, depending on whether the sum is larger than 24. (This is the case for Robot 1, 3, 4, 5, 7, and 8, as can be verified with the first Write statement.)
4. The expression \( \forall j \) \((\text{Hours}<8)\) returns true or false, depending on whether all values of \( H_{i,j} \) for a particular \( i \) are smaller than 8. (This is the case for Robot 3, 7, 9, and 10.)

5. The expression \( \exists j \) \((\text{Hours}=0)\) returns true or false, depending on whether there exists a value of \( H_{i,j} \) for a particular \( i \) that is zero. This is the case for all robots except 10.)

6. These three expressions combined form a subset defined as \( \text{il}(i) \). Hence LPL’s way to define subsets is indexing over a set

\[
\text{set } \text{il}(i);
\]

7. The operator \( \text{atleast} \) is another (Boolean) indexed operator, that returns true if at least a given number of Boolean expressions are true.

**Question**  (Answer see 20)

1. What is returned by: \( \max_i \) Price ?

2. What is returned by: \( \arg \min_i \) Price ?

3. What means: \( \max_i \) \( \min_j \) Hours ?

4. What means: \( \text{atleast}(5)\{i,j\} \) \((\text{Hours}>7)\) ?

**Answer**  (Question see 20)

1. 400. This is the largest value in the price list. Robot 9 costs 400.

2. 4. The smallest price value (50) is at the fourth position within the price list (Robot 4 costs 50).

3. It means “choose the smallest value in each column \( j \) of the matrix \( H_{i,j} \) and from the obtained list choose the largest value”. It is 4, because in step 1 the smallest value is 4.

4. It means that “at least five values in the table \( H_{i,j} \) are greater than 7”. True or false? This expression returns true (1).
21 Expression Evaluation (tutor13)

**Problem:** An LPL model does not need to be an optimisation model. Any sequence of declarations and instructions can be defined.

Listing 21: The Complete Model in LPL [1]

```lpl
model TUTOR13 "Expression Evaluation"
  /* I 'tutor.inc' */
  variable Robots(i);
  constraint
    Steps{j}: sum{i} Hours*Robots <= Capacity;
    Order{i}: Robots >= Ordered;
    C: Robots[9]=420;
  maximize revenue: sum{i} Price*Robots;
  parameter
    d{j}:=sum{i}Ordered*Hours;
    rCapa{j}:=Capacity-d;
    MaxR{i} := Floor(min{j|Hours} rCapa/Hours);
  Write(' Robot Ordered MaxR Total
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
        n' )
  Write{i}(' %-7s %3d %3d %4d
        i,Ordered,MaxR,MaxR+Ordered);
  Writep(revenue,Robots,Price,Ordered,Capacity,rCapa,Hours);
end
```

LPL Modeling Steps

The data are read from the file tutor.inc.

1. min and max are two index-operators which returns the minimum or maximum of a list of expressions.

2. The expression `sum{i}Ordered*Hours` returns for each step `j` the used capacity of the already Ordered robots. `rCapa{j}` is then the capacity that remains for the optimized production.

3. The parameter `MaxR{i}` calculates for each robot type `i` the maximally producible quantity of robots with respect to the capacities that is different from zero in the table `Hours` over `j`. Note that the condition “different from zero” is necessary in the definition otherwise it will be simply zero.

**Question** (Answer see 21)

1. Verify by solving the model that one cannot produce more than 520 units of robots of type 9.

2. Robot type 9 has the highest price. So it makes sense to produce the maximum possible – that is 520 – of it. What is the revenue, if one fixes the production of robot type 9 to 520?

3. Why does the previous fixing of variable 9 not generate the maximal revenue?

4. Simplify the expression `max{i} max{j} Hours[i,j]`. 
Answer (Question see 21)

1. Add a the following constraint

   \[
   \text{constraint C: Robots[9]} \geq 521;
   \]

   and solve the model. The model is infeasible (see message left below on the lp solver screen). The model has no solution because the capacity limits the production of robot of type 9 to 520.

2. The revenue is 222950. This is less than the maximally possible revenue of 232920.59.

3. The reason is not so easy to see. A production of 520 units of robots 9 would use all capacity in step 3. However, Robot 2 – which has the second highest price – does not use that much capacity in step 3. So maybe it would be worthwhile to leave capacity for robot 2 in order to get out more of the remaining capacities. That is indeed the case: We must reduce the production of Robot 9 to 420 to allow more production for robot 2, in order to increase revenue.

4. The two indices can be combined into one, One can use the second name H, and the index-list \([i, j]\) is not necessary. Hence, it is: \(\max\{i, j\}\) Hours.
Goal Programming (tutor14)

Problem: LPL can be used to model soft constraints using goal programming. The idea is – in our small example – that we do not know exactly the capacity of a process. That is, we want to integrate uncertainty, see explanation below.

Listing 22: The Complete Model in LPL [1]

```
model TUTOR14 "Goal Programming";
variable Marie; Jules; PTesting; Nrevenue; Prevenue; --slack variables
parameter dP:=18500 "desired revenue";
constraint
  Component: 5*Marie + 5*Jules <= 350;
  Mounting: 4*Marie + 8*Jules <= 480;
  Testing: 6*Marie + 2*Jules -PTesting <= 300;
  Order1: Marie >= 20;
  Order2: Jules >= 15;
  revenue: 300*Marie + 200*Jules -Prevenue+Nrevenue = dP;
  --TestBound: PTesting=10;
minimize deviation: Nrevenue + PTesting + Prevenue;
  --minimize deviation: Nrevenue + 200*PTesting + Prevenue;
Writep(deviation,Marie,Jules,Nrevenue,PTesting);
end
```

LPL Modeling Steps

Let’s return to the simplest model tutor02 with one exception.

1. Suppose we do not know exactly the capacity of Testing, but we know that it is at least 300.

2. Furthermore, we will be happy with a revenue of “about” 18500.

3. This problem would be infeasible as we know from problem tutor02 (since the maximizing revenue is 18000).

4. Hence, we add a positive slack variable (PTesting) in order to see how much the capacity of Testing has to be expanded exceeding 300 to attain our objective.

5. In the revenue constraint, we add two slack variables to cover a positive or negative deviation (Prevenue and Nrevenue).

6. We do not maximize revenue, as before, but we minimize a deviation measure.

7. The deviation of the positive PTesting and the negative Nrevenue slacks is minimized.

8. As can be seen from the solution, the optimum 18500 can be attained if we allow the Testing constraint to exceed its capacity of 300 by 20 units.

9. If we add an upper bound of 10 to PTesting (TestBound) then we can still attain a revenue of 18250.
**Question** (Answer see 22)

1. Modify the value of dP from 18500 to 20500 by steps of 200. What do you notice when solving each time?

2. Change the model in a way to make PTesting as small as possible.

**Answer** (Question see 22)

1. The revenue cannot be augmented over 19500, even if the "desired revenue" dP gets higher. This is because of the Components constraint is at 100% percent of its capacity and the number of Jules cannot be lowered below 15, because of the bound imposed by Order2 and consequently the number of Marie cannot be higher than 55. The next model tutor15.1pl shows how these calculations can be implemented in a single LPL model code.

2. We could replace the minimizing function to minimize dev: PTesting; . Another way is to leave the minimizing function, but to impose a higher "penalty" to the term PTesting by adding a factor, for example: 200*PTesting. Penalising is a general method to formulate multiple objectives.
23 Loop Programming (tutor15)

Problem: This model is the answer to the first question in the previous model tutor14. How to implement a sequence of optimisations in a single model? We call this also “parameterized optimisation”.

Listing 23: The Complete Model in LPL [1]

```lpl
model TUTOR15 "Loop Programming"
variable Marie; Jules;
  PTesting; Nrevenue; Prevenue; --slack variables
parameter dP:=18500 "desired revenue";
constraint
  Component: 5*Marie + 5*Jules <= 350;
  Mounting: 4*Marie + 8*Jules <= 480;
  Testing: 6*Marie + 2*Jules -PTesting <= 300;
  Order1: Marie >= 20;
  Order2: Jules >= 15;
  revenue: 300*Marie + 200*Jules -Prevenue+Nrevenue = dP;
  ---TestBound: PTesting=10;
-- loop
set i:=1..10;
parameter dP1{i}; Marie1{i}; Jules1{i}; revenue1{i};
for{i} do
  minimize deviation: Nrevenue + PTesting + Prevenue;
  dP := dP+200;
  dP1[i]:=dP; Marie1[i]:=Marie;
  Jules1[i]:=Jules; revenue1[i]:=300*Marie + 200*Jules;
end
Write(' dP Marie Jules revenue
  1.000 1000 1000 1000
... ... ... ...
  10.000 1000 1000 1000
');
end
```

LPL Modeling Steps

The only modification takes place at the minimize function. It is embedded in a loop: The model is solved 10 times each time with a different parameter dP.

1. First we add a set i. The cardinality of this set (here 10) is the number of optimisations.
2. for is another index operator that can be used to loop through a set, in our case over i.
3. Within the loop, the deviation is minimized and the parameter dP is increased in steps of 200.
4. Certain data are stored for later output and the loop is repeated.

Question (Answer see 23)


20
1. What happens if the capacity of Mounting is extended? Can we make more revenue?

2. We would like to rise $dP$ by 2\% within the loop. How can this be done?

**Answer** (Question see 23)

1. Nothing happens, the revenue is the same. This is because Mounting is never the limiting constraint. We would need to extend the Component capacity.

2. Replace the instruction $dP:=dP+200$; with $dP:=\text{Trunc}(1.02*\text{dP})$; and run the model again.
24 Logical Constraints (tutor16)

**Problem:** We include some logical constraints to our well know basic model tutor06\(^{21}\).

Listing 24: The Complete Model in LPL [1]

```plaintext
model TUTOR16 "Logical Constraints";
/*$1 'tutor.inc' */ — a data file is included here
integer variable Robots(i) [0..500];
constraint
  Steps{j}: sum(i) Hours*Robots <= Capacity;
  Order{i}: Robots >= Ordered;
  Log2: atleast(5) {i} (Robots[i]>=21);
maximize revenue: sum{i} Price*Robots;
Writep(revenue,Robots);
end
```

LPL Modeling Steps

The basic model (without the logical constraint Log1, Log2, Log3, and Log4) has the following solution:

revenue = 232850.00
Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7 Robot8 Robot9 Robot10
20 266 7 6 5 8 9 8 420 5

LPL allows to add Boolean operators within constraints. LPL translates these constraints automatically into mathematical linear constraints by introducing binary (0-1) variables. These feature allow the modeler to formulated certain requirements directly in a logical context.

Example: Suppose the following condition is required; “If the number of robot type 2 is larger or equal than 130 then at least 30 of robot type 6 should also be manufactured”. This is a logical implication formulated as:

\[
Robot_2 \geq 130 \rightarrow Robot_6 \geq 30
\]

In LPL, this can be formulated in a very similar way:

```plaintext
```

Adding this constraint to our basic model produces the following solution:

revenue = 227450.00
Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7 Robot8 Robot9 Robot10
20 254 7 6 5 30 9 8 407 5

We can verify that the condition is fulfilled: Robots[2] is larger than 130, therefore Robots[6] is at least 30 (which is the case in this solution). Of course, the revenue drop to 227450. That is the “price” the fulfill the condition.

LPL also allows one to use logical indexed operators. Suppose, we want to impose the following condition: “For at least 5 different type of robots, we need to manufacture a minimal quantity of 21”. In LPL, this can directly be formulated as follows:

\[ \text{atleast}(5) \{i\} \text{Robots}[i] \geq 21 \]

\(^{21}\) http://lpl.virtual-optima.com/lpl/Solver.jsp?name=tutor06
| constraint Log2: atleast(5){i}(Robots[i]>=21); |

Adding this constraint to the previous version (with Log1 produces the following optimal solution:

```
revenue = 225550.00
Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7 Robot8 Robot9 Robot10
21 231 7 6 5 30 9 8 405 21
```

One can easily verify that the condition Log2 also holds: At least 5 different robot types are manufactured at minimal quantity of 21. At the cost of a further reduction in the revenue.

Let’s add a third condition: “at least 10 units of robot 3 and at least 15 of robot 4 must be manufactured”. The condition in LPL is:


The optimal solution with the three first conditions becomes:

```
revenue = 221000.00
Robot1 Robot2 Robot3 Robot4 Robot5 Robot6 Robot7 Robot8 Robot9 Robot10
21 129 10 15 5 8 9 21 445 21
```

Note also that the condition Log1 is not used anymore: Because the number of robots of type 2 drops to 129 (which is smaller than 130) there is no need that the number of robot of type 6 is at least 30.

The condition Log3 can easily be implemented without any logical operators: it says simply that Robots[3] is bounded from below to 10 and Robots[4] has a lower bound of 15. The condition Log3 just emphasises simply that the Boolean and operator is the default operator of a list of constraint.

**Question**  (Answer see 24)

1. Formulate the constraint that none of the Robots type can exceed 100 pieces.

2. Formulate the condition: “If the number of robot type 1 is larger or equal than 10 then the number of robot type 5 must be larger or equal than 12 or the number of robot type 7 must be larger or equal than 10.”

**Answer**  (Question see 24)

1. A simple version is to set a lower bound for every variables as follows:

   ```
   constraint Log4{i}: Robots<=100;
   ```

   Another way to formulate it as a logical condition is:

   ```
   | constraint Log4: nor{i} (Robots>100);
   ```

   nor is another index operator which formulate the logical "none". Do not confound it with the operator nand. The operator nand – another operator in LPL – means “not all”. By the way, LPL translates the constraint Log4 automatically into ordinary variable bounds (see EQU-file).

2. The constraint in LPL is as follows:

   ```
   ```
Some Basic Expressions / Writes (learn01)

Problem: This code shows some simple operations and functions as well as Write formatted output using Write().

Listing 25: The Complete Model in LPL [1]

```lpl
model learn01 "Some Basic Expressions / Writes";
  parameter
    x := 5^7 "power operator";
    y := 1=1 or 0 and ~0 "Boolean expression";
    z1 := Sin(34) "Sinus";
    z2 := Log(1000) "log of base e";
    z3 := Max(34,6) "return the larger";
    z3a := Min(34,6) "return the smaller";
    z4 := 45/67*976 "arithmetic";
    z5 := 1<2 "Boolean";
    z6 := x "assignment";
    z7 := x/56 "item";
    z8 := y "Boolean assign";
  string parameter
    z9 := 'abcd' & 5 "String expression (concat)";
    z9 := '' & Sin(x) "cast double to string";
  date parameter zB :=@2018-12-08 "a date";
  Write('5^7 = %5d

1 or 0 and ~0 = %1d

Sin(34) = %6.4f

Log(1000) = %8.6f

Max(34,6) = %2d

Min(34,6) = %2d

45/67*976 = %7.3f

1<2 = %1d

x = %5d

x/56 = %8.4f

y = %1d

'abcd' & 5 = %s

'' & Sin(x) = %s

date as int = %d

da date format = %tc

, x, y, z1,z2,z3,z3a,z4,z5,z6,z7,z8,z9,zA,zB,zB);
end
```

65
26 Basic Indexed Expressions/Tables (learn01a)

Problem: This code shows some simple operations and functions.

Listing 26: The Complete Model in LPL [1]

```lpl
model learn01a "Basic Indexed Expressions/Tables";
  --- Tables expressions
  set i := [I1 I2 I3 I4 I5 I6 I7];
    j := [J1 J2 J3 J4 J5 J6 J7 J8 J9];
    k := [K1 K2 K3 K4 K5 K6 K7 K8 K9 K10];
  parameter a{i,j} := i*j;
    b{i,j} := i^2+j;
    c{j,k} := Sin(j)+Log(k);
  --- Matrix calculation
  parameter
d{i,j} := a[i,j] + b[i,j] "matrix addition";
  e{i,k} := sum{j} a[i,j] * c[j,k] "matrix multiplication";
  Writep(d,e);
  --- Random tables, sparse tables
  SetRandomSeed(10) "sets the random seed"
  parameter ra{i,j,k} := Rnd(10,20) "a 3-dimen. table";
    f{i,k} := sum{j} 10*ra[i,j,k] + if(e[i,k]>100, e[i,k] , 100);
  g{i,k|i<k} := 3 "sparse table";
  Writep(ra,f,g);
  --- 6. the first 50 fibonacci numbers
  set s := [ 1..50 ];
  parameter h{s}; H{s};;
    h[1]:=1 , h[2]:=2,
    {s|s>2} (h[s] := h[s-1]+h[s-2]);
    {s} ( H := if(s=1,1,s=2,2,H[s-1]+H[s-2])) "the same";
  Write(' FIBONACCI NUMER 1-50:
    Write{s}( '%2s %13d %13d
    Write(' %2s %13d %13d
  end
```

66
27 The Addm function (learn01b)

**Problem:** The function Addm() adds an element dynamically to a set, affecting all involved tables as well:

Listing 27: The Complete Model in LPL [1]

```lpl
model learn01b "The Addm function";
  set i := [I1 I2 I3 I4 I5 I6 I7];
  j := [J1 J2 J3 J4 J5];
  parameter a(i,j) := i*j;
  Writep(i,a);
  Addm(i,'ABC') "add dynamically an element to set i";
  Writep( a,i);
  {j} (a[#i,j]:=100+j) "fill the added slide";
  Writep(a,i);
  set k:=1..10;
  Addm(k,'HJK') "another example: add an element to set m";
end
```
Problem: This model shows further operators and functions of LPL.

Listing 28: The Complete Model in LPL [1]

```lplex
model learn02 "Some Math and Boolean Functions"

--- 1) predefined functions in LPL

set i := [1..6] ;

parameter

  a(i) := [ 8.99 8.01 -8.99 -8.01 8.5 -8.5 ];
  b(i) := [ 1.8 -8 4.01 7.35 9 -1.3 ];
  c(i) := [ 0 20 0 40 50 60 ];
  a1(i) := Ceil(a);
  a2(i) := Floor(a);
  a3(i) := Trunc(a);
  a4(i) := Abs(a);
  p(i) := i;
  maxi{i} := Max(a,b); -- which is larger?
  mini{i} := Min(a,b); -- which is smaller?
  ab{i} := Abs(if(Abs(a)>Abs(b) , a , b));
  ln_b{i} := Log(Abs(b));
  expo{i} := Exp(a/7);
  I_10_20{i} := Rnd(20,30); -- uniform distributed in the range [20,30]
  N_10_1{i} := Rndn(10,1); -- normal distributed, mean=10, std=1

Write(' a b Ceil Floor Trunc Abs Pos Max Min

Abs Ln(b) Exp Rnd Rndn
', a,b,a1,a2,a3,a4,p,maxi,mini,ab,ln_b,expo,
  I_10_20,N_10_1);

--- 2) Index-operators

parameter

  su := sum(i) b[i]; -- sum all b[i]
  pr := prod(i) b[i]; -- product
  ma := max(i) b[i]; -- the maximum
  mi := min(i) b[i]; -- the minimum
  pma := argmax(i) b[i]; -- position of the maximum
  pmi := argmin(i) b[i]; -- position of the minimum
  ex := exist(i) c[i]; -- is there any non-zero? (TRUE=1)
  fo1 := forall(i) c[i]; -- are there all non-zeros? (FALSE=0)
  fo2 := forall(i) b[i];
  atl3:= atleat(3)(i) c[i]; -- are at least 3 non-zero? (TRUE=1)
  atl4:= atmost(3)(i) c[i]; -- are at most 3 non-zero? (FALSE=0)
  atm := atmost(2)(i) (c[i]=0); -- are at most 2 equal to zero? (TRUE=1)
  exa := exactly(4)(i) c[i]; -- are exactly 4 non-zero? (TRUE=1)
  exa1:= exactly(3)(i) c[i]; -- are exactly 3 non-zero? (FALSE=0)

Write('results of all index-operators\n',
  'su,pr,ma,mi,pma,pmi,ex,fo1,fo2,atl3,atl4,atm,exa,exa1);

end
```

29 Logical Operators (learn02a)

**Problem:** This model shows further operators and functions of LPL.

Listing 29: The Complete Model in LPL [1]

```plaintext
model learn02a "Logical Operators";
set t := [1..4];
parameter x{t} := [1 1 0 0];  y{t} := [1 0 1 0];
Write('n');

    -------------------------------
    -- LOGICAL operators          
    -------------------------------
    x   y   |  and  or  NAND  nor  XOR  ->  <-  <->
    -------|--------|--------|--------|--------|--------|--------|--------|
              |        |        |        |        |        |        |        |

Write{t}(
    ' %1d %1d | %1d %1d %1d %1d %1d %1d
    %1d %1d
',
    x,y,x and y,x or y, ~(x and y), ~(x or y), x xor y, x->y, x<-y, x<->y);

--- Several examples of logical index operators
set q := [1..16];
binary parameter
    z{t,q} := [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0]
    b1{q} := and{t} z "same as forall{} ...";
    b2{q} := nand{t} z "same as ~and{}";
    b3{q} := or{t} z "same as exist{} ...";
    b4{q} := nor{t} z "same as ~or{}";
    b5{q} := xor{t} z "same as exactly(1){}";
    b6{q} := z[4,q] or xor{t} z;
    b7{q} := atleast(1) {t} z "same as or{}";
    b8{q} := atmost (2) {t} z;
    b9{q} := exactly (3) {t} z;
Write('n');
Write{q}(' %2s %d %d %d %d %d %d %d %d %d %d

end
```

69
30  The within and in Operators (learn03)

**Problem:** The keywords *within* and *in* are similar used in different contexts. (within was introduced in version LPL6.66 to differentiae two meanings.)

The first part of the model shows the *within* in an expression: \( i \) *within* \( j \) returns the position of element \( i \) within set \( j \), zero if not found (note that the first position is 1).

The second part shows it in a indexed part as \{...\}. For example, \( i \) *in* \( s \),... introduces a *local* identifier \( i \) as a place-holder for \( s \) in the following part of the expression.


```lpl
model learn03 "The within and in Operators";
    −−−− first example with the within-operator −−−−
    set i := [1..12];
    set j := [2..7];
    parameter x{i}:=i;
    r1{i | i within j} := x;
    r2{i | ~(i within j)} := x;
    r3{j} := x[j within i];
    r4{i}:= x[i within j];
    Writep(x,r1,r2,r3,r4);
    −−−− second example with in-operator −−−−
    set s := [s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11];
    t := [s2 s3 s4 s5 s12 ];
    parameter a{i in s,j in s | i<7 and (j=i+1 or j=i+2 or j=i+5)}:=1 "Compares position";
    b{s,t | t within s} :=1 "Compares the element names";
    Writep(a,b);
end
```

---
31 Some String Compare Operations (learn04)

**Problem:** String comparision: string can be used as litterals within ' ' or as identifiers declared as string parameter. They can be used anywhere in an expression, where a number or another identifier is allowed. In general they return a zero if evaluated in an expression except in the following cases:

1. used in a applied index-list in place of an element. In this case it is tested if this element exists. If not, zero is returned. example: ... + x['apple','period4'] + ...

2. together with one of the six relational operators. (<, <=, =, >, >=, <> ) example: 
   
Listing 31: The Complete Model in LPL [1]

```lpl
model learn04 "Some String Compare Operations"; 
set       i := [ KA EI FE KH ET FS ]; 
parameter a(i) := [ 1 2 3 4 5 6 ]; 
b := a['EI'] "reference element by name"; 
b1 := a[2] "same but reference by position"; 
c := a['KA']; 
c1 := a[1]; 
d0 := '100' < '15' "true"; 
d1 := 100 < 15 "false"; 
e(i) := i > 'FS' "string comparison!"; 
f(i) := i >= 'FS'; 
g(i) := i < 'FS'; 
h(i) := i <= 'FS'; 
j(i) := i <> 'FS'; 
k(i) := i = 'FS'; 
string parameter s := ['Fette']; 
t(i) := [ 'Kalorien', 'Eiweiss', 'Fett', 'Kohlenhydrate', ' tierische Eiweisse', 'Fett sichtbar' ]; 
parameter m(i) := t <= t[3]; 
m1{i} := t < t['FS']; 
n(i) := i < t; 
o{i} := s < t; 
Writep(m); --- check others 
end
```
32 Index options, and wrap around function (learn05)

Problem: Note $p$ and $t$ are two basic sets that have nothing in common, although one can test whether an element from one is also in the other with the in operator. However, $t1$ is a subset (a relation) of $p$. Normally, it does not make sense to mix two basic sets. See what happens here by generating the EQU file (locally).

Listing 32: The Complete Model in LPL [1]

```
model learn05 "Index options, and wrap around function";
set
  p := [0..14] ;
  t := [1..13] ;
  t1{p} := p<>'0' and p<>'14';
variable X{p};
constraint
  x0: sum{p} X;
  x1: sum{p} X[p];
  x3: sum{i in p} X[i];
  y0: sum{t} X[t within p];
  y1: sum{t1} X[t1];
  y2: sum{i in t} X[i];
  w0: sum{i in p} X[i-1];
  w1: sum{t1} X[t1-1];
  w2: sum{i in t1} X[i];
  v0: sum{i in p} X[(i+#i-3)%#i+1];
  v1: sum{i in p} X[(i+#i-3)%#i+1];
solve;
```

Another small example
```
set i := [1..5]; k{i} := [1 3 5];
parameter a{k} := [10 30 50];
parameter b{i} := k;
parameter c{i} := a;
Writep(i,k,a,b,c);
end
```
33  Sort function (learn06)

**Problem:** This example shows how to sort a vector \( a_i \). The vector \( a \) is not sorted itself. The sort generates a permutation vector \( b \) of the same length as \( a \) and the sorted vector \( a \) can then be generated by the syntax \( a[b] \) (or more explicit \( a[b[i]] \)).

Listing 33: The Complete Model in LPL [1]

```lpl
model learn06 "Sort function";
  set i:=1..10;
  parameter a{i}:=Trunc(Rnd(1,100));
  integer b{i};
  Sort(a,b);
  Writep(a,b);
  Write{i}('%3d \t %5.2f \n',b,a[b]);

  set j:=1..5;
  parameter c{i,j}:=Trunc(Rnd(1,100));
  d{i,j};
  Sort(c,d);
  Writep(c,d);
  Write{i,j}('%3d \t %5.2f \n',d,c[d]);
end
```

73
### 34 Date/Time Type (**learn07**)

**Problem:** Date parameters are defined as given in the model (see `datetime`). Note that internally a date is treated as a double and can be treated as such.

Output a date is done using the `%t...` formatting. One can also use the `Format()` function for generating a string of the date.

Listing 34: The Complete Model in LPL [1]

```lpl
model learn07 "Date/Time Type";
    date parameter datetime:=@2018-12-11T13:10:15;
    Write('A date and time: %tc , %tr\n', datetime, datetime);
    string parameter dt:=Format('%tc %tT', datetime, datetime);
    Write('dt=%s\n\n', dt);
    date parameter birthday := @1962-02-26;
    Write('Your birthday is : %tF\n', birthday);
    Write('You already live %d days.\n\n', Now()-birthday);
    set i:=1..30;
    Write{i}('At: %tF is your %5d-th birthday\n',
               birthday+1000*i, 1000*i);
end
```
Documenting Models (learn08)

Problem: This is a model with inline documentation text. The model documentation can be generated automatically using a compiler switch (T and t) (see reference manual).

This is the documentation for the main model. The model is a simple linear program with 10 variables. All data are declared and defined inline, that is, they are defined within the model code.

Modeling Steps

Note that the code contains comments for each entity. In lplw.exe, press key F5 to generate the documentation. Make sure that the batch file modeldoc.bat contains the correct paths. ..... This is the LPL code:

Listing 35: The Complete Model in LPL [1]

```lpl
model learn08 "Documenting Models";
/**
 * First a set @i@ is defined. It has 10 elements.
 * It is used to define the 10 variables.
 */
set i := [ 1..10 ] "A set with 10 elements";
/**
 * A variable vector @x@ defined the model variable.
 * it is indexed over @i@ (as $x_i$).
 */
variable x{i} "The number of different type of robots";
/**
 * The following parameters define model data
 */
parameter HC{i} := [ 5 5 4 5 6 5 7 8 4 7 ] "Component time";
HM{i} := [ 4 8 5 6 4 4 7 6 5 3 ] "Mounting time";
HT{i} := [ 6 2 4 6 3 4 5 2 5 3 ] "Testing time";
Ordered{i} := [ 20 15 7 6 5 8 9 8 7 5 ] "Quantity ordered";
Price{i} := [ 300 200 100 50 50 100 200 100 400 200 ];
/**
 * Next, four model constraints are defined, three of them are
 * capacity restrictions, and the last one is a demand requirement.
 */
constraint
    Component : sum{i} HC[{i}] * x[i] <= 3500 "Component building";
    Mounting: sum{i} HM[{i}] * x[i] <= 4800 "Mounting robots";
    Testing:  sum{i} HT[{i}] * x[i] <= 3000 "Testing robots";
    Order{i}:  x[i] >= Ordered[i] "Ordered";
/**
 * Finally, we define the objective function, which is to
 * maximize the revenue (or total sell).
 */
maximize revenue: sum{i} Price[i]*x[i] "Maximize the profit";
/**
 * Data and the result are finally written to a file,
 * the so called NOM-file. It has the same name as the
 * model file (here @doc.lpl@) with extension 'nom'.
 * Hence, the results are written to the file @doc.nom@.
 */
Writep(revenue,x,HC,HM,HT);
/** We also can define submodels within main models. The next
```
The Model

**function** learn08

First a set \( i \) is defined. It has 10 elements. It is used to define the 10 variables.

```plaintext
set i "A set with 10 elements"

A variable vector \( x \) defined the model variable. it is indexed over \( i \) (as \( x_i \)).

```plaintext
var x_i "The number of different type of robots"
```plaintext

The following parameters define model data

```plaintext
param HC_i "Component time"
param HM_i "Mounting time"
param HT_i "Testing time"
param Ordered_i "Quantity ordered"
param Price_i
```plaintext

Next, four model constraints are defined, three of them are capacity restrictions, and the last one is a demand requirement.

```plaintext
s.t. Component : \( \sum_i HC_i \cdot x_i \leq 3500 \) "Component building"

s.t. Mounting : \( \sum_i HM_i \cdot x_i \leq 4800 \) "Mounting robots"

s.t. Testing : \( \sum_i HT_i \cdot x_i \leq 3000 \) "Testing robots"

s.t. Order_i : \( x_i \geq Ordered_i \) "Ordered"
```plaintext

Finally, we define the objective function, which is to maximize the revenue (or total sell).

**maximize** revenue := \( \sum_i Price_i \cdot x_i \) "Maximize the profit"

Data and the result are finally written to a file, the so called NOM-file. It has the same name as the model file (here doc.lpl) with extension 'nom'. Hence, the results are written to the file doc.nom.

**Writep(revenue, x, HC, HM, HT)**

We also can define submodels within main models. The next declaration is such a submodel, called submodel. However It is not used. It is only for documentation purpose.

```plaintext
function submodel

one may declare other entities here...

set i

end
```
Problem: This model shows the effect of the CALLBACK option for the solver (Gurobi).
(Note that the correct Gurobi version is installed!).

First it defines the solver interface string gurobiLSol (see manual Chap 9.2) This is necessary for the very last parameter (5000), which is the SIP22 - meaning 20secs. The solver is called with the option 'CALLBACK=callit' which means that the solver must call the submodel callit on a regular base. The first time the solver calls it is when it gets its first feasible solution. Then it calls it each time when a new integer feasible solution is found but not before 20secs have elapsed.

This option allows the modeller to save intermediary solutions for a lengthy calculation of a very hard IP problem. Try out different value for the SIP22. The default is 0secs, which means that the submodel is called inside the solver each time when a new integer feasible solution has been discovered.

In our case, the callit just adds a line to the NOM-file when it was called.

Listing 36: The Complete Model in LPL [1]

```lpl
model learn10 "Call Submodel within Solver (Gurobi)";
string parameter gurobiLSol := ',,lib:C:/gurobi811/win64/bin/
gurobi811,gurobi.prm,,TimeLimit=100;OutputFlag=0,,LP;MIP;
QP;iQP;QCP;iQCP,,5000';
SetSolver(gurobiLSol,'CALLBACK=callit');
set i,j:=1..16; s:=1..15;
parameter c{i,j,s}:=Trunc(Rnd(1,30));
binary variable x{i,j,s|i<>j};
constraint
  A{i,j|i<j}: sum{s} (x[i,j,s]+x[j,i,s]) = 1;
  B{i,s}: sum{j} (x[i,j,s]+x[j,i,s]) = 1;
minimize obj: sum{i,j,s} c*x;
model callit;
  parameter i;;
  i:=i+1;
  Write('obj = %2d after %d secs\n', sum{i,j,s} c*x,GetParam(1)/1000);
end
end
```

77
37  Reading Relations (learn12)

Problem: How to read “relations” from text files.

Listing 37: The Complete Model in LPL [1]

```lpl
model learn12 "Reading Relations";
   -- define three basic sets
   set i; j; k;
   -- define four relations
   set ij{i,j}; ik{i,k}; jk{j,k}; ijk{i,j,k};
   parameter a{i,j,k};
   -- reading them from file
   Read(i,j,k)('learn12.txt', i,j,k,ij,ik,jk,ijk,a);
   Writep(i,j,k,ij,ik,jk,ijk,a);
end
```

Note: reading relations does not physical read in text files. So in fact the Read instruction only reads i, j, and k, (and a). The 4 relations are derived. This is a powerful option to generate sparse relations.

The content of the text file 'learn12.lpl' is as follows:

```
// a data set (i,j,k,a) for learn12.lpl
1 a A 12
1 b A 13
2 b B 15
3 a A 17
3 c C 19
4 a A 21
4 b A 11
4 b B 8
5 a B 7
5 b A -3
5 c C 33
```
Multiple bounds of variables (learn14)

Problem: Bounds can be assigned by the bound attribute \([a..b]\) at the declaration of an entity. Bounds on variables can also be assigned by constraints. Generate the EQU-file to see the result.

Listing 38: The Complete Model in LPL [1]

```lpl
model learn14 "Multiple bounds of variables";
set i := [1..10];
parameter a{i} := i;
b{i} := 10+i;
variable
  x [1..99]  "x has lower/upper bound of 1 and 99";
y{i} [a..b]  "The lower/upper bound of y is given by the vectors a and b";
z{i} [i..10+i]  "Any expression can be used as a bound";
constraint
  r1: x <= 100  "This constraint will have no effect, upper bound is still 99";
r2: x >= 2  "This overrides the lower bound of 1";
r3{i}: y <= 15  "Some upper bounds of y are reassigned";
r4{i}: z >= 4  "Some lower bounds of z are reassigned";
maximize obj1: x + sum{i}(y+z)  "The solution is at the upper bounds";
Writep(x,y,z);
minimize obj2: x + sum{i}(y+z)  "The solution is at the lower bounds";
Writep(x,y,z);
end
```
39 Submodels and Encapsulation (learn15)

Problem: This model declares four (sub)models (m1-m4). The executable part of the main model runs the four models one after the other. Note that the same identifier can be redeclared in different models.

Listing 39: The Complete Model in LPL [1]
model m4 "fourth model";

set
  i := [ A1 A2 A3 A4 A5 A6 A7 A8 A9 ]; — a basic set
  j(i) := [ A2 A3 A4 A5 A6 A7 A8 A9 ]; — a derived subset of set i
  k(i) := [ A3 A4 A5 A6 A7 A8 ]; — a derived subset of set i
  p(i) := [ A4 A5 A6 ]; — a derived subset of set i
  m := [ A3 A4 A5 A6 A7 A8 ]; — m is an independent basic set
  — note: m has nothing in common with any of the above sets!

parameter
  simple_i(i) := i; — there is nothing special about these expressions
  simple_j(j) := j;
  simple_k(k) := k;
  simple_p(p) := p;
  /* the complication enters only, if the sets are mixed. Consider the following expressions */
  a(i) := 1000*i;
  b(i) := 1000*i + 100*j;
  c(i) := 1000*i + 100*j + 10*k;
  d(i) := 1000*i + 100*j + 10*k + p;
  /* you should read them as
   a{i} := 1000*i;
   b{i} := 1000*i + 100*j[i]; — if i=1 then j[i] is zero.
   c{i} := 1000*i + 100*j[i] + 10*k[i];
   d{i} := 1000*i + 100*j[i] + 10*k[i] + p[i];
   which is also legal syntax in LPL. */
  e(i) := i within m; — the in operator
  f(m) := m within i;
    /* Note: "e{i} := m" (or f{m}=i) would produce an error, since m cannot
    be bound to i because they are two independent basic sets. But "m in i" returns the position of m within i; it is 0, if a corresponding m is not in i. */

Write('writes from the model m4\n');
Writep(i,j,k,p,m);
Writep(simple_i,simple_j,simple_k,simple_p,a,b,c,d,e,f);
end
end
40  Again, some non-trivial relations (learn16)

Problem: A example with relations

Listing 40: The Complete Model in LPL [1]

```lp
model learn16 "Again, some non-trivial relations";

--- a first example: compare 'ge0' and 'ge1' ---
set    i := [1..10];
j := [1..10];
k := [1..2];
s(i,j) := [ 5 5, 5 7, 5 9 ] "a 2-dimen. sparse relation";

string parameter
  typ(k,s) := [ '<' '=' '>' '<' '=' '<' '>' ] "a 3-dimen table, note: cardinality is 6";

parameter
  ge0(k,i,j | typ[k,s[i,j]]='<') := 100*k+10*i+j;
  gel(k,s[i,j] | typ[k,s]='<') := 100*k+10*i+j;
/* both, ge0 and gel, return the same values. But there is a subtile difference: ge0 has a cardinality of 200, gel has a cardinality of only 6. Therefore, gel is more compact and much quicker to work through.
   Note also, that i and j in gel are locals, the globals i and j are not 'visible'. In ge0, however, i and j are globals. */

--- another example ---
set    ii,p,q := [I1 I2 I3 I4 I5 I6 I7 I8 I9 I10];
jj := [J1 J2 J3 J4 J5 J6];
kk(ii) := [ I5 I6 I7 ];
mm(jj) := [ J2 J4 J6 ];

parameter
  a1(p,q | kk[p] and mm[q]) := 1;
a2(kk,mm) := 1;
Write('This model outputs nothing, run it locally\n');
end
```

82
41 Expressions and Constraints (learn17)

**Problem:** Active constraints are model constraints that are passed to the solver; inactive entities are not passed to the solver. A way to select active constraints is through subject to: A list of active constraints is given by (id, id, ...) A list of inactive constraints is given by (id, id, id, ...).

Part of a constraint can be declared in 'expression'.

Listing 41: The Complete Model in LPL [1]

```lpl
model learn17 "Expressions and Constraints";
variable X; Y; Z;
expression
  B:= 10;
  C:= 11;
  A:= B+C;
  D:= X+Y;
constraint
  E : D - Z = 3;
  F : D + D + Z = A;
  G : 3*D + 2*Z >= A+10;
minimize obj1: X - Y subject_to E,F;
Write('OBJ1=%3d , X=%4.1f , Y=%4.1f , Z=%4.1f\n', X-Y,X,Y,Z);
minimize obj2: X - Y subject_to learn17,~G;   --same as before
Write('OBJ2=%3d , X=%4.1f , Y=%4.1f , Z=%4.1f\n', X-Y,X,Y,Z);
minimize obj3: X - 2*Y subject_to G,F;
Write('OBJ3=%3d , X=%4.1f , Y=%4.1f , Z=%4.1f\n', X-2*Y,X,Y,Z);
end
```
### 42 GetValue Function (learn20)

**Problem:** The function `GetValue()` returns various values of an entity. If `x` is a variable then `GetValue(x, 3)` returns the dual value of `x` (if available).

#### Listing 42: The Complete Model in LPL [1]

```plaintext
model learn20 "GetValue Function";
    set i := [1..10]; j := [1..9];
    parameter A{i,j} := if (Rnd(0,1)<0.25, Rnd(0,5));
        c{j} := Trunc(Rnd(10,100));
        b{i} := Trunc(if (Rnd(0,1)<0.75, Rnd(0,20)));
    variable x{j} [1..50];
    constraint R{i} : sum{j} A*x >= b;
    minimize obj : sum{j} c*x;
    Write(’j = %6s
’,{j} j);
    Write(’Value of x = %6.1f
’,{j} x);
    Write(’GetValue(x,0) = %6.1f
’,{j} GetValue(x,0));
    Write(’GetValue(x,1) = %6.1f
’,{j} GetValue(x,1));
    Write(’GetValue(x,2) = %6.1f
’,{j} GetValue(x,2));
    Write(’GetValue(x,3) = %6.1f
’,{j} GetValue(x,3));
    Write(’GetValue(R,3) = %6.1f
’,{i} GetValue(R,3));
    Write(’GetValue(R,4) = %6.1f
’,{i} GetValue(R,4));
    Write(’GetValue(R,5) = %6.1f
’,{i} GetValue(R,5));
    Write(’GetValue(R,6) = %6.1f
’,{i} GetValue(R,6));
    Write(’GetValue(R,7) = %6.1f
’,{i} GetValue(R,7));
    Write(’GetValue(R,8) = %6.1f
’,{i} GetValue(R,8));
end
```

84
Problem: Run this model locally with the interpreter option ‘q’. This will generate two files: ‘learn21.sql’ and ‘learn21.sql2’. The first is a complete executable SQL script that can be used to create a database. The second file is the corresponding read/write listing in LPL syntax able to read write to/from that database.

Listing 43: The Complete Model in LPL [1]

```
model learn21 "create a SQL script, test sparcity (locally only)";
  -- create the SQL-script and compare it to the LPL model
  set i; j;
  ij{i,j} := [ i1 j1, i2 j2, i3 j3, i3 j4];
  parameter
    aij{i,j} := / i1 j2 1, i2 j3 2, i3 j4 3, i3 j1 4, i2 j1 5/;
    bij{i,j} := / i1 j1 10, i2 j3 20, i2 j4 30, i3 j2 40/;
end
```
44 GetAttr Function (tow parameters) (learn22)

Problem: Implements reflection! The function GetAttr() returns a statement attribute (see chap 4.1 in the reference manual. For example, if $X$ is an integer variable then GetAttr($X, 2$) returns the string 'integer'.

Listing 44: The Complete Model in LPL [1]

```lpl
model learn22 "GetAttr Function (tow parameters)"
  // GetAttr with TWO parameters
  set i := [i1 i2 i3 i4 i5 i6 i7 i8 i9 i10];
  parameter a{i}:=i*10;
  integer variable X,a1,a2{i|a>=50}
    default 20
    priority i+2
  [i..i^2]
  "Comment"
    'a quote'
  frozen
    := 100+i^4;
  Write('GetAttr(a,3) = %s

GetAttr(X,2) = %s

GetAttr(X,3) = %s

GetAttr(X,4) = %s

GetAttr(X,5) = %s

GetAttr(X,7) = %s

GetAttr(X,8) = %s

GetAttr(X,9) = %s

GetAttr(X,10) = %s

GetAttr(X,12) = %s

GetAttr(X,13) = %s

GetAttr(X,14) = %s

GetAttr(X,15) = %s

GetAttr(X,16) = %s

GetAttr(X,18) = %s

GetAttr(X,19) = %s

GetAttr(X,20) = %s

GetAttr(X,22) = %s

GetAttr(X,23) =%\n
GetAttr(X,24) =%\n
GetAttr(X,25) =%\n
GetAttr(X,26) =%\n
GetAttr(X,2),GetAttr(X,3),GetAttr(X,4),GetAttr(X,5),GetAttr(X,7),
GetAttr(X,8),GetAttr(X,9),GetAttr(X,10),GetAttr(X,12),
GetAttr(X,13),GetAttr(X,14),GetAttr(X,15),GetAttr(X,16),
GetAttr(X,18),GetAttr(X,19),GetAttr(X,20),GetAttr(X,22),
GetAttr(X,23),GetAttr(X,24),GetAttr(X,25),GetAttr(X,26));
end
```
45  GetAttr Function (one parameter) (learn22a)

Problem: The GetAttr() function can also be called with a single parameter. In this case it returns the statement attribute of the focused entity. (see learn2222).

Listing 45: The Complete Model in LPL [1]

```plaintext
model learn22a "GetAttr Function (one parameter)";
// GetAttr with ONE parameters
set i := [i1 i2 i3 i4 i5 i6 i7 i8 i9 i10];
parameter a{i} := i*10;
integer variable
X,a1,a2{i|a>=50}
  default 20
  priority i+2
  "Comment"
  'a quote'
  frozen
  := 100+i^4;
SetFocus(a);
Write('GetAttr(3) (focus is a) = %s\n',GetAttr(3));
SetFocus(X);
Write('GetAttr(2) = %s\n
GetAttr(3) = %s',
  GetAttr(2),GetAttr(3));
end
```

46 GetName Function (learn23)

**Problem:** The GetName function returns the name of an instanced entity in various formats. Example: if \( X \) is defined as in the model below then \( \text{GetName}(X[1,2],0) \) returns ‘\( X[i1,B] \)’ and so on. Run the model and check.

Listing 46: The Complete Model in LPL [1]

```lpl
data learn23 "GetName Function";
  set i := [i1 i2];
    j := [A, B, C];
  string parameter I{i}:=['firstI', 'secondI'];
    J{j}:=['1stJ', '2ndJ', '3rdJ'];
  parameter I{i} := [100 200];
  StringToSet(i,I); — link i to I
  StringToSet(j,J); — link j to I //try this
  variable X{i,j}:=1;
  Write{i,j}('GetName(X,0) = %s\n',GetName(X,0));
  Write{i,j}('GetName(X,1) = %s\n',GetName(X,1));
  Write{i,j}('GetName(X,2) = %s\n',GetName(X,2));
  Write{i,j}('GetName(X,3) = %s\n',GetName(X,3));
  Write{i,j}('GetName(X,4) = %s\n',GetName(X,4));
  Write{i,j}('GetName(X,5) = %s\n',GetName(X,5));
end
```
GetParams Function (learn24)

**Problem:** The function `GetParams()` returns a "global" parameter of the model. For example, `GetParams(8)` returns the actual LPL version you are running, etc.

Listing 47: The Complete Model in LPL [1]

```lpl
model learn24 "GetParams Function";
   Write('GetParams(0) = %s
',GetParamS(0));
   Write('GetParams(1) = %s
',GetParamS(1));
   Write('GetParams(2) = %s
',GetParamS(2));
   Write('GetParams(3) = %s
',GetParamS(3));
   Write('GetParams(4) = %s
',GetParamS(4));
   Write('GetParams(5) = %s
',GetParamS(5));
   Write('GetParams(6) = %s
',GetParamS(6));
   Write('GetParams(7) = %s
',GetParamS(7));
   Write('GetParams(8) = %s
',GetParamS(8));
   Write('GetParams(9) = %s
',GetParamS(9));
   Write('GetParams(10) = %s
',GetParamS(10));
   Write('GetParams(11) = %s
',GetParamS(11));
   Write('GetParams(12) = %s
',GetParamS(12));
   Write('GetParams(13) = %s
',GetParamS(13));
   Write('GetParams(15) = %s
',GetParamS(15));
   Write('GetParams(16) = %s
',GetParamS(16));
   Write('GetParams(17) = %s
',GetParamS(17));
end
```

89
48 Split Function (learn25)

**Problem:** The function Split splits (or tokenizes) a string into several parts separates by a character (the splitting character). For example, if the string is ‘aa,bb,ccc,d’ and the splitting parameter is , (a comma), then the function can tokenize it into ‘aa’, ‘bb’, ‘ccc’, and ‘d’. The result must be a parameter. Example:

```plaintext
string a; b; c; d;
Split(‘aa,bb,ccc,d’,,a,b,c,d);
```

The string parameters a, b, c, and d receive the parts as string. If the parts are numbers (or dates) then a type cast is automatically done. Example:

```plaintext
integer a; b; c; d;
Split(‘12,1234,1,45’,,,a,b,c,d);
```

The parameters a, b, c, and d will have the numerical values 12, 1234, 1, and 45.

Another form of the Split function to tokenize a (list of) strings into a single string table. Example:

```plaintext
string parameter S{i} := [‘a,b,c,d,e,f’ ‘1,2,3’ ‘AA,BB,CC,DD’];
set k := [1..7];
string parameter A{i,k};
{i}Split(S,,,(k) A);
```

In this case the string parameter A(i,k) will receive all string parts split. Run the model to see the effects. Such a split may be especially interesting in a Read statement.

Listing 48: The Complete Model in LPL [1]
49 String Functions (learn26)

**Problem**: Note there is no type cast function that transforms a number into a string. To transform a number into a string use the concatenation operator & with an empty string, such as ’’&12.3 which returns a string ’12.3’.

Listing 49: The Complete Model in LPL [1]

```lpl
model learn26 "String Functions";
parameter d := Strdate('2018-12-20');
parameter d1:= Strdate('1899-12-31');
parameter r := Strfloat('56.17');
parameter t := Strfloat('1,2.3,4.5,5\',\',\',3); 参数 a := Strlen('abcd');
parameter b := Strpos('bc\',\'abcdef');
string parameter c := Strreplace('abcedf\',\'bc\',\'-ccbbbc-');
string parameter s := Strsub('abcdefg\',\',3); Write(d1,r,t,a,b,c,s);
Write('%s\n','Length of\'abcd\' is ' & a);
Write('%tc\n', d);
end
```

To transform a string into a number type, use the function Strfloat. This function can also extract the n-th occurrence of a substring. For example the parameter t in the model was extracted from the 3-th substring, separated by the comma character (,).
50  String Operations and Format (learn26a)

**Problem:** Show some string operations and the Format function

Listing 50: The Complete Model in LPL [1]

```lpl
model learn26a "String Operations and Format";
    set i:=[1..5]; q:=[1..4];
    set IQ{i,q}:=if (Rnd(0,1)<=.5,1,0);
    string parameter x{i,q} := Format(‘i%1sq%1s’, i, q);
    string parameter concatB{q} :=’’;
    for{i} do concatB{q|IQ}:=concatB & x[i,q] & ’’; end
    string parameter concatB1{q}:=Format(’%4s%1s’, {i|IQ} x,’’);
    Writep(concatB,concatB1);
    Write(’\n’);
    string parameter a{i} := [’abcdef’, ’12345c789’, ’qwcffc’, ’gggggh’, ‘cuitrtzuiop’];
    integer parameter lenA{i} := Strlen(a);
    integer parameter posC{i} := Strpos(’c’,a);
    string parameter subA{i} := Strsub(a,3,20);
    string parameter formA := Format(’Total: %8.2f %6s %7d Date: %tc’,
    234.567, ’Uiii’, 456, Now());
    string parameter formB := Format{i} (’I:%2s 	i^4=%3d 
’, i,i^4);
    Writep(lenA,posC,subA);
    Write(’\n/%s/\n\n’, formA);
    Write(formB);
    Write{i} (’I:%2s 	i^4=%3d 
’,i,i^4);
end
```

51 Functions NextFocus, NextPosition (learn27)

Problem: This model shows the use of the functions NextFocus and NextPosition. The first part of the model is a small production/distribution model containing three set and two variable declaration.

The goal is to output all sets and the corresponding elements. This is done in the second part: First the focus is set to the beginning of the model by calling SetFocus(). Then a loop is repeated as long as NextFocus(0) returns true (not zero). The function jumps to the next basic set (0 is the parameter value for genus 0 - which is the basic set. If there is no more basic set then the function NextFocus(0) returns false (0) and the loop ends. The inner loop jumps from one data entry to the next in lexicographical ordering. In this case, it traverses all elements of the focused set. The function NextPosition returns true as long as a next entry exists, otherwise it returns false (0). Note that we use GetValueS() to return the element name, because it is a string.

The third part outputs all variables with their value and dual value. The mechanism is the same as in the second part. In this case however the function NextFocus(3) jumps from a variable entry to the next (3 is the parameter for genus 3 - that is, the variable. Note that in this case we must use GetValue() to retrieve the values because they are numerical.

Listing 51: The Complete Model in LPL [1]

```lpl
model learn27 "Functions NextFocus, NextPosition";
set s := [ SS HD ];
d := [ BA PH WA RI ];
p := [ P1 P2 P3 ];
parameter trcost{s,d} := [ 3.52 9.47 0.38 8.63, 2.04 6.61 7.22 9.97 ];
prcost{s,p} := [ 4.22 5.05 4.60, 1.45 2.45 2.03 ];
prCap{s} := [ 5000 6000 ];
orders{d} := [ 120 100 30 234 ];
variable PR{p,s}; SH{p,s,d};
constraint Capacity{s} : sum{p} PR <= prCap;
Balance{p,s} : PR = sum{d} SH;
Demand{p,d} : sum{s} SH >= orders;
minimize costs: sum{p,s} (prcost*PR + sum{d} trcost*SH);
Write(’LIST OF ALL BASIC SETS\n’);
SetFocus();
while NextFocus(0) do
  Write(’%s %s (alias: %s)\n’, GetAttr(3),GetAttr(4),GetAttr(12));
  while NextPosition do Write(’%s\n’, GetValueS(0)); end
end
Write(’\nLIST OF ALL VARIABLES\n’);
SetFocus(0);
while NextFocus(3) do
  Write(’%s %s\n’, GetAttr(3), GetAttr(4));
  while NextPosition do Write(’%s = %5.2f (dual=%4.2f)\n’,
    GetName(0),GetValue(0),GetValue(3));
end
end
```
52 Multiple Snapshots (learn29)

Problem: This model shows the use of multiple snapshots. A snapshot file is a compact file that stores all the data (variable values included) from a model into a file. It is therefore at the moment of writing a ‘snapshot’ of the actual state. The instruction:

```plaintext
write to ‘<filename>.sps’;
```

generates automatically such a file. All you need is, that the file extension must be sps.

Listing 52: The Complete Model in LPL [1]

```plaintext
code
model learn29 "Multiple Snapshots"
  set i; j; parameter a{i,j};
  test1;
  test2;
  test3;
  test4;
  model test1;
    i:=[A B D]; j:=[a b c d];
    a{i,j}:= [1 2 3 4 5 6 7 8 9 10 11 12];
    Write(’1.sps’);
  end
model test2;
  i:=[A C B]; j:=[a b c d];
  a{i,j}:= [1 2 3 4 5 6 7 8 9 10 11 12];
  a{i,j}:=10*a;
  Write(’2.sps’);
end
model test3;
  i:=[D G A]; j:=[b c d a];
  a{i,j}:= [1 2 3 4 5 6 7 8 9 10 11 12];
  a{i,j}:=100*a;
  Write(’3.sps’);
end
model test4;
  Read(’1.sps’);
  Read(’:+:2.sps’);
  Read(’:+:3.sps’);
  Writep(a);
end
end
code
```
53 Show Graph.Component (learn30)

Problem: This model shows the use of the function Graph.Component. The model is an assignment problem, augmented with subtour elimination constraints of length 2. Run the model and open the SVG graph learn30.svg.

The model also shows how to separate the main model from the data and from the output code. The model data and output are called automatically at an appropriate time of execution (data is called just before the minimization, and output is called at the very end). The user can change this by calling them explicitly at the intended place.

Note also that output is a friend of model data which means that the entities defined inside data are also available within the model output without explicitly noting the dot-notation (data.x for example).

Listing 53: The Model in LPL [1]

```lpl
model learn30 "Show Graph.Component";
parameter n:=30 "Graph Size";
set i,j,k "A set of vertices";
parameter c{i,j} "Distance between two locations i and j";
parameter cNr1 "number of components";
    passed "loop variable";
    C{i} "the components";
binary variable x{i,j|i<>j} "Is 1 if (i,j) is in the tour";
constraint
    A{i}: sum{j} x = 1;
    B{j}: sum{i} x = 1;
    S2{i,j|j>i}: x[i,j] + x[j,i] <= 1; // no subtours of length 2
minimize obj: sum{i,j} c*x;
cNr1:=Graph.Components(x,C);
Write('There are %d components
',cNr1);
while passed<cNr1 do
    passed:=passed+1;
    Write('
Component %d: ',passed);
    Write{i|C=passed}('%2s ',i);
end
end
```

Listing 54: The Data Model

```lpl
model data;
    parameter X{i}; Y{i}; m:=Trunc(Sqrt(n));;
i:=1..n;
    X{i}:=(i%m+1)*2+Trunc(Rnd(0,2));
    Y{i}:=(i/m+1)*2+Trunc(Rnd(0,2));
    c{i,j}:= Sqrt((X[j]-X[i])^2+(Y[j]-Y[i])^2);
end
```

Listing 55: The Output Model

```lpl
model output friend data;
    Draw.Scale(30,30);
    Draw.DefFont('Verdana',8);
    for(i,j|x) do Draw.Line(X[i],Y[i],X[j],Y[j],3); end;
    for(i) do Draw.Circle(i&'',X,Y,.3,1,0) ; end;
    Draw.Text('Length='&Round(obj,-4),0,0,16);
end
```
54 Show Graph.Mincut function (learn31)

Problem: The function Graph.Mincut() implements the Stoer-Wagner min cut algorithm. Unfortunately, this function has not yet been well tested, so no guarantee is given... (to be checked)

Listing 56: The Complete Model in LPL [1]

```lpl
model learn31 "Show Graph.Mincut function";
    set i,j;   --nodes
e{i,j};   --edges
    y{i,j}; YY{i};
    parameter X{i}; Y{i};
    parameter u{i,j}; C;

readgraph;
C:=Graph.Mincut(u,y,YY,0);
Write('Min Cut value is: %d\n', C);
Write{i,j|y}(' (%1s,%1s)\n', i,j);
Write{i|YY}(' %s ', i);
drawgraph;
model readgraph;
    i:=[1..8];
    u{i,j}:=[1 2 2, 1 5 3, 2 3 3, 2 5 2, 2 6 2, 3 4 4, 3 7 2,
        4 7 2, 4 8 2, 5 6 3, 6 7 1, 7 8 3];
    u{i,j}:=[u,u,u[i,j]];
    e{i,j}:=[i<j,u];
    X{i}:=[0 1 2 3 0 1 2 3];
    Y{i}:=[0 0 0 0 1 1 1 1];
end
model drawgraph;
    e[2,1]:=0;  //do nor draw
    Draw.Scale(100,100);
    Draw.DefFont('Verdana',8);
    for{e[i,j]} do Draw.Arrow('(&y&','&u&')',
        X[i],Y[i],X[j],Y[j],.2,if(y=u,3,y>0,4,0)); end
    for{i} do Draw.Circle(i&'',X,Y,.2,1,0); end
end
end
```
Problem: The function Graph.MStree() implements the Prim algorithm for the minimal spanning tree problem in a graph. (Note that the graph data are read for THIS file(!) directly.)

Listing 57: The Complete Model in LPL [1]

```lpl
model learn32 "Show Graph.MStree (Minimal Spanning Tree)"
    set i,j,k; e(i,j); x(i,j);
    parameter X(i); Y(i);
    parameter c(i,j|e); len;
    readgraph;
    c(i,j):=Sqrt((X[j]-X[i])^2+(Y[j]-Y[i])^2);
    len:=Graph.Mstree(c,x);
    drawgraph;
model readgraph;
    string parameter FILE:=[‘learn32.lpl’];
    Read{i}(FILE&’,%1:Coordinates’, i,X,Y);
    Read{i,j}('%1:Edges’, i,j,e,c);
    e(i,j):=if(e,e,e[j,i]);
end
model drawgraph;
    e[2,1]:=0;
    Draw.Scale(60,60);
    Draw.DefFont(‘Verdana’,8);
    for{e[i,j]|x} do Draw.Line(Round(c,-2)&’’,X[i],Y[i],X[j],Y[j],3);
end
    for(i) do Draw.Circle(i&’’,X,Y,.2,1,0); end
end
```

Read Multiple Snapshots (learn33)

Problem: This model is a small example that shows the power of snapshot files. A snapshot file is a file that records ALL model data at the call of a Write snapshot instruction. Later on, this snapshot can be read in. The filename has extension sps. The following function call simply writes a snapshot file named snap.sps.

\[
\text{Write('snap.sps');}
\]

The instruction writes a snapshot of the complete internal data store of the LPL model at the moment of its call to a file - the snapshot file, that is, all data at a particular moment of the execution are collected and written to the file. This “data state” can be restored at any time by a snapshot Read instruction.

\[
\text{Read('snap.sps');}
\]

Note that data tables with the frozen attribute set are not modified.

Listing 58: The Complete Model in LPL [1]

```lpl
model learn33 "Read Multiple Snapshots";
set i,j frozen :=[A B C D E]; —unchangable set
set k :=[1..4];
parameter a; b{i}; c{k,i};
for {i} do
  a:=10*i; —assign a value to a
  b{j}:=i*j; —fill table b
  c[k,j]:=j+10+k+i; —fill table c
  Write(i&'.sps'); —write snapshot files: A.sns, B.sns, ...
end;
ClearData(learn33);
Read('D.sps');
Write('\nRead the snapshot D.sps only:\n________________________\n');
Writep(a,b,c);
ClearData(learn33);
Write('\nRead all snapshots cummulatively:\n________________________\n');
for {i} do
  Read('+:'&i&'.sps'); —read snapshot files cumulative
end;
Writep(a,b,c);
end
```

Snapshot files are particular interesting, when a lengthy optimization takes place: After the optimization, a snapshot can be written. At a later date, this snapshot can be read again and LPL’s data store is in a state as if the optimization had taken place.

Several snapshots can be written to various files, as is done in the model above within the first loop. What is interesting: all the snapshot can be read individually or cumulatively. Cumulatively means, that

1. The LPL internal store is augmented with an additional index-set called _SNAP_.
2. All data tables are expanded by the new index-set, that is, a singleton becomes indexed, a vector becomes a two-dimensional table, etc.
3. The “slices” of the tables are cumulatively filled by each read snapshot.
4. At the end, we have all snapshots stored in parallel and the tables can be viewed as if they were ordinary tables - augmented by the snapshot index-set.
This allows the modeller to compare the different variants.

In the first loop of the model, the parameter $a$ is modified to become: 10, 20, ..., 50, the tables $b$ and $c$ are filled with different values and the snapshot files $A.sps ... E.sps$ are written.

Then all data are cleared, with the exception of the set $i$ which has the attribute `frozen'. Then just one particular snapshot is read - the fourth ($D.sps$). And the data are written to control LPL's data.

The data are again cleared and in the second loop, all snapshot files are read with the prefix reading instruction `+:', which means to read the snapshots cumulatively. All data are again written to view LPL's internal store.
57 Freeze function (learn34)

**Problem:** This model is a small example that shows the use of the indexed Freeze function. “Freezing” a variable means to fix its preset value and they are sent as fixed bounds to the solver. Of course, LPL can change these values at any time using an assignment.

“Freezing a constraint” means to drop a constraint such that the solver does not see it. Variables and constraints can be “frozen” and “unfrozen” at any time using Freeze and Unfreeze.

Listing 59: The Complete Model in LPL [1]

```plaintext
model learn34 "Freeze function";
set
i := [1..4];
j := [1..5];
c1(i) := i <= 3;;
binary variable
x[i,j];
p_C1{c1[i]};;
p_C1{c1[i]} := 0; // Assign three variables
Freeze({i|i=1} p_C1[i]); // Freeze variable p_C1[1] to 0
constraint
C1{c1[i]} : sum(j) x[i,j] <= 1 + p_C1[i];
Freeze({i|i=2} C1[i]); // Freeze a constraint
maximize Obj: sum{i,j} x[i,j];
Write('Obj: %f
', Obj);
end
```

The model above defines 3 variables `p_C1` for the first three indexes in `i`. The following instruction fixes the first variable to zero:

```
Freeze({i|i=1} p_C1[i]);
```

This variable is sent to the solver as a fixed-bounded variable.

The model also declares three constraints `C1` of which the second is “frozen” with the instruction

```
Freeze({i|i=2} C1[i]);
```

To see the effect of these two instructions, the user must generate the EQU-file. LPL only generates the following constraints

```
C1[1]:  - p_C1[1] + x[1,1] + x[1,2] + x[1,3] + x[1,4] + x[1,5] <= 1;
```

One can also see that the value of `p_C1[1]` is 0 as it was frozen to that value.
Problem: All elements of a set in LPL have a "dual nature". The elements are first of all strings, even if some have numerical values. But since all sets in LPL are ordered, each element also has a position. The first element of a set has position 1, the second has position 2, and so on.

.... to be checked ...

Listing 60: The Complete Model in LPL [1]

```plaintext
model learn35 "function En(), El()";
  set i:=[Aa Bb Cc Dd Ee Ff];
  Write('set i is: %3s
', {i} GetValueS(i,0));
  Write('The third element in i is: %s
', En(i,3));
  Write('Dd is at position %d in set i
', El(i,'Dd'));
  Write{i}('%2d
',El(i,i));
end
```

101
Show Graph.Bfs (learn36)

**Problem:** This model shows the use of the function Graph.Bfs() which implements the depth first search of a graph. The instruction in the model Graph.Bfs(e,p,0); explores the component starting at node '0'. The search tree is the subgraph with the red edges. A more interesting example is given in model maze23.

Listing 61: The Complete Model in LPL [1]

```lpl
model learn36 "Show Graph.Bfs";
  parameter n:=[20] "Graph Size";
  set i,j "A set of vertices";
    e(i,j) "A edge list";
    t(i,j) "tree edges";
  parameter p{i} "the search tree";
  Graph.Bfs(e,p,0);
  {i|p[i]<>-1} (t[i,p[i]]:=1, t[p[i],i]:=1);
model data;
  parameter X{i}; Y{i}; m:=Trunc(Sqrt(n));
  i:=0..n-1;
  X[i]:=(i%m+1)*2+Trunc(Rnd(0,2));
  Y[i]:=(i/m+1)*2+Trunc(Rnd(0,2));
  {i,j} if(Rnd(0,1)<=.03, (e[i,j]:=1, e[j,i]:=1));
end
model output friend data;
  Draw.Scale(30,30);
  Draw.DefFont('Verdana',8);
  {e[i,j]} Draw.Line(X[i],Y[i],X[j],Y[j]);
  {t[i,j]} Draw.Line(X[i],Y[i],X[j],Y[j],3,2);
  {i} Draw.Circle(i&'',X,Y,.2,1,0);
end
end
```

60 Show Graph.SPath (shortest path) (learn37)

Problem: The function Graph.SPath() implement the Belman-Ford algorithm for the shortest path problem.

Listing 62: The Complete Model in LPL [1]
Problem: The function $Sl(a, b)$ can be used to model "soft" constraints (goal programming), that is, it introduces slack variables automatically and penalizes them in the objective function. "$a$" is an expression that models the upper bound of the slack variable (the lower bound is 0). "$b$" is an expression for the weight in the objective function. To explain the different usages, this model gives same hints.

Listing 63: The Complete Model in LPL [1]

```lpl
model learn39 "S1 function (goal programming)"
set i:=1..3; j:=1..5; k:=1..3;
variable x{i}; y{i}; z{j}; w[0..0]; v;
parameter a{i}:=10*i; b{i}:=100*i;
parameter cc{i,k}:=k; ww{i,k}:=10*k;
constraint A{i}: w+x+y+sum{j} z >= 1 - Sl(a>11,a) - Sl(3,b);
constraint B{i|i<3}: x >= 2*i - Sl(2,0.9);
constraint C{i|i>1}: sum{j} z >= 1 - Sl{k|k<=2}(cc,ww);
constraint D{i|i>1}: sum{j} z >= 1 - sum{k|k<=2} Sl(cc,ww);
minimize obj: sum{i} x;
Write('The slacks of B are: %5.1f\n', {i|i<3} GetValue(B,9));
Write('Because: The values of the left hand side (x) are 0 2, the right hand side are: 2 4\n');
end
```

In the constraint $A$ the function $Sl(a>11,10*i)$ is used. It adds a new indexed variable to the model as follows:

```
| variable A_x[i] [0..a>11];
```

The variable name is the corresponding constraint name plus '_x'. The upper bounds of these variables are $a[i]>11$ (which is 0 for $a_1$ and 1 for $a_2$ and for $a_3$ in this case). Slack variables with an upper bound of 0 (since the lower bound is also 0) are eliminated automatically from the model. (Note: other variables with a lower and upper bound of zero, like $w$ are not eliminated from the model.) In addition, the following terms are added to the objective function $obj$:

```
| ... + 20*A_x[2] + 30*A_x[3] + ...
```

Also in the constraint $A$ a second function $Sl(3,b)$ is used. It adds a second slack variable to the model with the fixed upper bound of 3 and a weight of $b_i$ to the objective function:

```
| variable A_x1[i] [0..3]
```

The variable name is the corresponding constraint name plus '_x1'. (If several $Sl$ functions occur in a constraint, then the corresponding additional slack variables name contains an additional number 1, 2, ... ) It also adds the following terms to the objective function:

```
| ... +100*A_x1[1] +200*A_x1[2] +300*A_x1[3]
```

The constraint $B$ displays the sparsity aspect. Since $B$ is only defined for $i \leq 2$ the corresponding slack variables $B_x[i]$ are also generated with the same sparsity.

The constraint $C$ and $D$ are exactly the same. The $Sl$-function can be indexed and this is nothing else than "syntax-sugar" and is translated into the constraint $D$ where the index-list is removed from the $Sl$ and placed in a sum operator. The slack variables are indexed correspondingly. In the model case, the variable definitions are:

\[^{24}\text{Note that it makes perfectly sense in real applications to used several slacks with different upper bounds and penalties (weights) in the objectives. This allows the modeller to model piece-wise linear penalties for various goals.}\]
The function GetValue(B, 9) returns the value of the corresponding slack variable. Note also that the compiler switch 'p' adds a $Sl$ function term to each constraint. LPL generates the following EQU-file (equation listing):

\[
\begin{align*}
\text{min: } & +10 \ D_x[1,2] +10 \ D_x[1,3] +20 \ D_x[2,2] +20 \ D_x[2,3] +10 \ C_x[1,2] +10 \ C_x[1,3] \\
& +20 \ C_x[2,2] +20 \ C_x[2,3] +0.9 \ B_x[1] +0.9 \ B_x[2] +100 \ A_x[1] +200 \ A_x[2] \\
\end{align*}
\]


Note: the $Sl$ function can also be added in an objective function, in this case, it implements a preemptive goal programming, see model library\textsuperscript{25} and compare it with model library-1\textsuperscript{26}.

\textsuperscript{25} http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/library
\textsuperscript{26} http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/library-1
Parameterized Calling of Submodels (learn40)

**Problem:** This model shows how one can generated several model variants using the `subject_to` attribute. The variants depend on the parameter `var`. For example, if `var='case2'` then the model first runs the submodel `case2` then it continues running the main model which minimizes a function subjected to the constraints defined in main model `learn40` and in addition in the submodel `addConst0`.

Listing 64: The Complete Model in LPL [1]

```plaintext
model learn40 "Parameterized Calling of Submodels";
    var:= 'case2',
    if(var='case1', case1,
        var='case2', case2,
        var='case3', case3,
        var='case4', case4);
    // or alternatively: RunMain:
    ----- end main run: all declarations begin here
    ----- note: declarations can be in ANY order
    string parameter var; // parameter to choose "variant"
set i;
parameter a{i}; c{i}; B;
integer variable x{i};
constraint C: sum{i} a*x = B;
minimize obj: sum{i} c*x
subject_to
    if(var='case1',learn40,
        var='case2',(learn40,addConst0),
        var='case3',(learn40,addConst0),
        var='case4',learn40);
    Writep(x);
/* alternatively: this main model to run
model RunMain:
    var:= 'case2';
    if var='case1' then case1; output; end;
    if var='case2' then case2; end;
    if var='case3' then case3; end;
    if var='case4' then case4; end;
end */
----- the model "variants"
model case1;
    myData0; aPlus10;
end
model case2;
    Writep(var);
    myData; addConst0;
end
model case3;
    myData; aPlus10; addConst0;
end
model case4;
    myData; minItems;
end
----- several variations
model myData0;
```

106
i := [1..10];
a{i} := [3 4 5 2 6 7 4 5 3 9];
c{i} := [2 3 5 7 9 8 6 4 2 1];
B := 233;
end

model myData "A model data (defining the base case)"
  i := [ chocolate milk apple water mais chips tomatoes beer wine tabacco ];
a{i} := [3 4 5 2 6 7 4 5 3 9];
c{i} := [2 3 5 7 9 8 6 4 2 1];
B := 233;
end

model aPlus10 "Augment a by 1"
  a{i} := a+1;
end

model addConst0 "Exactly 30 items"
  constraint A: sum(i) x = 30;
end

model minItems "Minimize the number of items"
  Freeze(obj); //replace objective function !!!
  minimize minI: sum(i) x subject_to learn40;
end

model output;
  Write('Capacity is: %d
', B);
end
end
Problem: A model in LPL is like a function — by the way model and function are exchangeable keywords — that can be called and can return a value. The first example is mymodel, a parameterless model that returns 4.

```plaintext
model mymodel;
    return 2*2;
end
```

Models definition can also contain formal parameters like isSlotEarly. This model calls itself other functions and returns a Boolean value. Note that recursive calls are not possible in LPL.

```plaintext
function isSlotEarly(integer _d1; _d2);
    return isSlotE(_d1, _d2) or isSlotSE(_d1, _d2) or isSlotE1(_d1, _d2);
end
```

A Third example is the function Euclid which returns the greatest common divisor of two positive integers.

```plaintext
model Euclid(integer a;b);
    integer t;
    return while (b > 0, (t := a, a := b, b := t % b, a));
end
```

Common to all these functions is that the formal parameters must be declared as parameters and are singleton (not indexed). In this case, the parameters are passed as value (not as reference). Model learn4227 shows an example where parameters are passed by reference.

Listing 65: The Complete Model in LPL [1]

```plaintext
model learn41 "A Model with Parameters I";
    parameter e_StartE := 10; e_EndE := 100;
    parameter e_StartSE := 11; e_EndSE := 90;
    parameter e_StartE1 := 12; e_EndE1 := 80;
    function isSlotE(integer _d1; _d2);
        return _d1 >= e_StartE and _d2 <= e_EndE and _d1 <= _d2;
    end
    function isSlotSE(integer _d1; _d2);
        return _d1 >= e_StartE and _d2 <= e_EndE and _d1 <= _d2;
    end
    function isSlotE1(integer _d1; _d2);
        return _d1 >= e_StartE and _d2 <= e_EndE and _d1 <= _d2;
    end
    function isSlotEarly(integer _d1; _d2);
        return isSlotE(_d1, _d2) or isSlotSE(_d1, _d2) or isSlotE1(_d1, _d2);
    end
model Euclid(integer a;b);
    integer t;
    return while (b > 0, (t := a, a := b, b := t % b, a));
end
model mymodel;
    return 2*2;
end
Write('`\mymodel` called, returns : %d
', mymodel);
Write('%b
', isSlotEarly(20, 110));
```

27 http://lpl.virtual-optima.com/lpl/Solver.jsp?name=/learn42
Write('gcd of 994009 and 96709 is : %d
', Euclid(994009, 96709));
Problem: This model shows submodels (functions) with arguments. It contains two submodels mult2 that returns a value multiplied by 2. The formal parameter is an expression or a singleton parameter. In this case, they are passed by value.

The second submodel knapsack is a complete knapsack model with 5 formal parameters being passed. It encapsulates a complete optimisation model and return the objective value. 4 of the five parameters are passed by reference and one is passed by value (K) because it is a singleton parameter. When calling the model the parameters must exactly match: a set must be called as a set, a variable must be called as a variable, even the type must match: If a formal parameter is defined as an integer variable then it must also be called as an integer variable.

The third example is the model returnAsIs. The argument x is a singleton value, so it must be passed by value. However the call of the function contains a indexed parameter, but since it is passed as A[j] it is considered as an expression and hence pass by value.

Listing 66: The Complete Model in LPL [1]
Problem: Download the model and open the 4-dimensional table $a$ in lplw.exe and play around.

Listing 67: The Complete Model in LPL [1]
**Problem:** (This model supposes that you understand how FastReport works.) This model shows a report that colors the cells depending on their values: positive value are blue, negative values are red.

The first Write creates a new database and adds a table `titleP` with the unique field 'title' and a unique row with the entry 'Titel'. Furthermore it will create two ReportBands: a `TfrxReportTitle` with a unique `TfrxMemoView` named `titleP_titel`, and a `TfrxPageFooter` (Option '0').

The second Write adds a database table named `tableH` with 5 fields named b, a1,a2,a3,a4. Their content is '' (a blank), and the elements of p (2007 2008 2009 2010). Furthermore, it adds a ReportBand `TfrxPageHeader` with the corresponding fields.

The third Write adds a table `ReportColor` and adds 9 fields. It will also add a ReportBand `TfrxMasterData`. Finally it generates the report template `ReportColor.fr3` (if it does not exist). Then it creates a pdf file of the entire report.

If the report template `ReportColor.fr3` has been created then add the following code to the code page of `ReportColor.fr3`:

```pascal
function rgb(r,g,b:int):int;
var c:int;
begin c:=trunc(b); c:=c shl 8 + trunc(g); c:=c shl 8 + trunc(r);
  result:=c;
end;

function col(Sender: TfrxMemoView) : int;
var d,a: double; x:int;
begin
  d := 150;
  a := Sender.Value;
  if (a<=511) and (a>=-511) then x:=trunc(d*abs(a)) else x:=511;
  if a<0 then begin if x>255 then x:=rgb(511-x,0,0) else x:=rgb(255,255-x,255-x); end
  else begin if x>255 then x:=rgb(0,0,511-x) else x:=rgb(255-x,255-x,255); end;
  //if (a<0) then Sender.Font.Color:=clWhite;
  Result:=x;
end;

procedure EvalOnAfterData(Sender: TfrxMemoView);
begin
  Sender.Color:=col(Sender);
end;

Then dd the function `EvalOnAfterData` as an event `OnAfterData` to the four MemoViews `ReportColor_a1 .. ReportColor_a4`.
```
Circular Time Lag Operator (learn45)

**Problem:** Since a set in LPL is always ordered, we can define circular relations (first element follows the last one).

Listing 68: The Complete Model in LPL [1]

```plaintext
model learn45 "Circular Time Lag Operator";
set i "periods" := [1 2 3 4 5 6 7 8 9 10 11 12];
parameter s{i} "number of desks open in a period"
   := [ 10 10 8 8 14 14 5 5 3 3 8 8 ];
t{i} :=
   [ 9.2 8.8 8.8 8.4 8.0 8.0 8.4 8.8 8.8 9.2 9.6 9.6 ];
variable x{i} "number of clerks beginning in period i"
constraint r{i}: x + x[i#i+1] + x[(i+2)#i+1] + x[(i+3)#i+1] >= s;
minimize z: sum{i} t*x;
Writep(z,x);
end
```
Function $x^P$ (learn46)
Problem:

Listing 69: The Complete Model in LPL [1]

```plaintext
model xDrawAll "Draw Lines/Opacity"

Draw.DefFont('Verdana,sans-serif',13,-2,2);
Draw.DefFill('rect',Rgb(0,255,0));
Draw.DefFill('circle',Rgb(255,255,51),Rgb(180,180,180),10);
Draw.DefFill('circle.d',-1,-2,1);
Draw.DefFill('path',Rgb(0,255,0),0);
Draw.DefFill('path.d',-1,Rgb(0,255,0),10);
Draw.Line('path.e',-2,-2,-2,-2,3);
Draw.Line('path.f',-2,-2,-2,-2,2);
Draw.Line('path.g',-2,-2,-2,-2,1);
Draw.Line('line.x',-1,Rgb(0,255,0),20);
Draw.Line('line.d',-1,Rgb(0,255,0),2);
Draw.Line('line.r',-2,-2,-2,2);
Draw.Line('line.s',-2,-2,-2,3);
Draw.DefLine('line.x',-1,Rgb(0,255,0),20);
set i:=[1..5];

parameter o1[i]:=[1 .2 .8 .5 1];
o2[i]:=[1 .2 .8 1 .5];

string t[i]:=['normal','opacity .2','opacity .8','fill-opacity .5','stroke-opacity.5'];

for {i in 1..5} do
  Draw.Rect(20+140*(i-1),30,120,60);
  Draw.Circle(80+140*(i-1),60,50,-2,-2,-2,o1,o2);
  Draw.Text(t,22+140*(i-1),130);
end;

set j:=[1..3]; string parameter t1[j]:=['the form','fill-rule:nonzero ','fill-rule:evenodd';

parameter px[i]:=[120,-90,30,30,-90];
py[i]:=[0,100,-70,70,-100];

for {j in 1..3} do
  Draw.Path('M',20+200*(j-1),150);
  for {i} do Draw.Path('l',px,py); end;
  Draw.Path('Y',if (j=1,-1,-2),-2,-2,-2,-2,if (j=3,2,-2));
  Draw.Text(t1,100+200*(j-1),210);
end;

Draw.Line('#x',30,400,30,300);
Draw.Line('#r x',80,400,80,300); //stroke-linecap:round
Draw.Line('#s x',130,400,130,300); //stroke-linecap:square
Draw.Line(10,300,163,300,0,1);
Draw.Line(10,400,163,400,0,1);

Draw.Text('stroke-linecap',20,430);

Draw.Text('butt',15,340);
Draw.Text('round',65,360);
Draw.Text('square',115,380);

Draw.Path('M',180,400); Draw.Path('l',120,-50); Draw.Path('l ' 'Y',70,100);

Draw.Path('#d e','Y');

Draw.Text('stroke-linejoin:round',240,440);

Draw.Path('M',220,370); Draw.Path('l',120,-50); Draw.Path('l ' 'Y',70,100);

Draw.Text('butt',15,340);

Draw.Text('square',115,380);
```

115
Figure 10: Output
70 Use arc path (xDrawArcPath)

Problem:

Listing 70: The Complete Model in LPL [1]

```
model xDrawArcPath "Use arc path";
  Draw.Scale(2,2);
  --Draw.Scale(2,-2);
  Draw.Path('M',20,255);
  Draw.Path('L',100,255);
  Draw.Path('A',10,70,90,1,1,180,255);
  Draw.Path('A',70,100,70,1,0,260,255);
  Draw.Path('L',370,255);
  Draw.Path('Y',-1,0);
  Draw.Circle(20,255,3);
  Draw.Circle(100,255,3);
  Draw.Circle(180,255,3);
  Draw.Circle(260,255,3);
  Draw.Circle(370,255,3);
end
```

Output as follows:

![Figure 11: Output](imageurl)
71 Drawing Circles (**xDrawCircle**)  

**Problem:**

Listing 71: The Complete Model in LPL [1]

```lpl
model xDrawCircle "Drawing Circles";
  −− Draw.Scale(3,3);
  Draw.Scale(2,-5);
  Draw.Circle(15,15,10,2,3,5);
  Draw.Circle(‘0.5’,40,40,10,1,0);
  Draw.Ellipse(‘12.0’,60,0,30,20,2,3,4,0.2,0.2);
  Draw.DefLine(‘dash’,8,3,0);
  Draw.Ellipse(‘#dash’,’14.0’,120,50,40,50,2,3,4);
  Draw.Triangle(0,80,20);
end
```

Output as follows:

![Output Diagram]

Figure 12: Output
72 A Quadratic Constraint (xDrawCircle1)

**Problem**: This is a small model with a (convex) quadratic constraint. Note that this model can only be solved with particular solvers such as Cplex12. Therefore we force the use of this solver using the LPL function `SetSolver(...)`. The submodel `draw` produces the picture in Figure 13.

![Figure 13: The Solution](image)

The model is as follows:

```
model xDrawCircle1 "A Quadratic Constraint";
    SetSolver(knitroLSol);
variable x; y; v; w;
constraint A: x^2 + y^2 <= 1;
maximize obj: x+y;
Writep(x,y);
draw;
model draw;
    Draw.Scale(100,100);
    Draw.Circle(2,2,1,-1,0);
    Draw.Line(2,0,2,4);
    Draw.Line(0,2,4,2);
    Draw.Line(2,0.58,3.42,2,3,3);
    Draw.Circle(x+2,2-y,.04);
    Draw.Text('(x,y) = ('&Round(x,-2)&' , &Round(y,-2)&')',2+x,1.8-y);
end
end
```

Listing 72: The Complete Model in LPL [1]
73  Show Color Numbers in LPL (\texttt{xDrawColors})

**Problem:** Colors assignment in LPL to positive integers.

**Listing 73: The Complete Model in LPL [1]**

```lpl
model xDrawColors "Show Color Numbers in LPL";
set y:=[1..8] "Eight color tables";
x:=[1..32] "32 darkness in gradients";
integer parameter a(y,x):= 32*(y-1)+x-1;
integer parameter b(y,x):= 32*(y-1)+32-x;
Draw.Scale(30,30);
Draw.Line(1,1,2,1,1);
Draw.Text('The color numbers 0 to 255 in LPL are as follows:',2,2,30);
{y,x} Draw.Rect((32*(y-1)+x-1)&'',x,y+2,1,1,a);
--- using RGB function for colors.
{y,x} Draw.Rect((i-1)/8,12,1/8,2,Rgb(0,0,i-1),Rgb(0,0,i-1));
{y,x} Draw.Rect((i-1)/8,15,1/8,2,Rgb(i-1,i-1,255),Rgb(i-1,i-1,-1,255));
end
```

Output as follows:

The color numbers 0 to 255 in LPL are as follows:

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 32| 33| 34| 35| 36| 37| 38| 39| 40| 41| 42| 43| 44| 45| 46| 47| 48| 49| 50| 51| 52| 53| 54| 55| 56| 57| 58| 59| 60| 61| 62| 63| 64| 65| 66| 67|
| 64| 65| 66| 67| 68| 69| 70| 71| 72| 73| 74| 75| 76| 77| 78| 79| 80| 81| 82| 83| 84| 85| 86| 87| 88| 89| 90| 91| 92| 93| 94| 95| 96| 97| 98|
| 96| 97| 98| 99|100|101|102|103|104|105|106|107|108|109|110|111|112|113|114|115|116|117|118|119|120|121|122|123|124|125|126|127|128|129|
|128|129|130|131|132|133|134|135|136|137|138|139|140|141|142|143|144|145|146|147|148|149|150|151|152|153|154|155|156|157|158|159|160|161|162|
|160|161|162|163|164|165|166|167|168|169|170|171|172|173|174|175|176|177|178|179|180|181|182|183|184|185|186|187|188|189|190|191|192|193|
|192|193|194|195|196|197|198|199|200|201|202|203|204|205|206|207|208|209|210|211|212|213|214|215|216|217|218|219|220|221|222|223|224|225|226|
|224|225|226|227|228|229|230|231|232|233|234|235|236|237|238|239|240|241|242|243|244|245|246|247|248|249|250|251|252|253|254|255|256|

Figure 14: Output


74 Draw Filters (xDrawFilters)

**Problem:** This model example shows SVG filters.

Listing 74: The Complete Model in LPL [1]

```lpl
model xDrawFilters "Draw Filters";
    Draw.DefFilter('f', 9,11,1, 4);   // 9=feGaussianBlur
    Draw.DefFilter('f',13,1 ,2, 4,4); //13=feOffset
    Draw.DefFilter('f',14,1 ,3, 5, .75,20,-2,Rgb(187,187,187));
        //14=feSpecularLighting
    Draw.DefFilter('f',18 ,-5000,-10000,20000);   //18=fePointLight
    Draw.DefFilter('f', 4,3,11,3,2);    // 4=feComposite
    Draw.DefFilter('f', 4,10,3,4,6,0,1,1,0);   // 4=feComposite
    Draw.DefFilter('f',11);                   //11=feMerge
    Draw.DefFilter('f',24,2);                //24=feMergeNode
    Draw.DefFilter('f',24,4);                //24=feMergeNode
    Draw.Rect(1,20,198,85,Rgb(136,136,136),4);
        //Draw.Group('#f');
    Draw.Path('#f','M50,90 C0,90 0,30 50,30 L150,30 C200,30 200,90 150,90 z',-1,5,10);
    Draw.Path('#f','M60,80 C30,80 30,40 60,40 L140,40 C170,40 170,80 140,80 z',5);
        //Draw.EndGroup();
    Draw.Text('#f','LPL',62,76,40);
    Draw.Rect(1,125,198,60,Rgb(136,136,136),4);
    Draw.Circle('#f',30,155,25,2);
    Draw.Rect('#f',70,130,50,50,3);
    Draw.Path('#f','M130,155 150,-25 10,50 z',4);
end
```

Output as follows:

![Output Diagram](image_url)

Figure 15: Output

121
Show radial gradient \textbf{(xDrawGrad0)}

\begin{figure}[h]
\centering
\includegraphics[width=\linewidth]{radialGradient.png}
\caption{Output}
\end{figure}

\textbf{Problem:}

Listing 75: The Complete Model in LPL [1]

```
model xDrawGrad0 "Show radial gradient";
  Draw.DefFont('Verdana, sans-serif', 50, -2, -2, 2);
  parameter blue:=5; yellow:=6; red:=3;
  Draw.DefGrad('rgr1', 1,.5,.5,.5, .5,.5, 0,blue,1,yellow);
  Draw.DefGrad('rgr2', 1,.9,.5,.9, .9,.5, 0,blue,1,yellow);
  Draw.DefGrad('rgr3', 1,.3,.7,.7, .3,.7, 0,blue,1,yellow);
  Draw.DefGrad('rgr4', 1,.5,.5,.5, .5,.5, 0,blue,.5,yellow,1,blue);
  Draw.DefGrad('rgr5', 1,.5,.5,.5, 1,.5, 0,blue,1,yellow);
  Draw.DefGrad('rgr6', 1,.5,.5,.5, .5,1, 0,blue,1,yellow);
  Draw.DefGrad('rgr7', 1,.8,.3,.9, .8,.3, 0,blue,1,yellow);
  Draw.DefGrad('rgr8', 1,.5, 1,.9, .3,0, 0,blue,1,yellow);
  Draw.DefGrad('rgr9', 1,.5,.5,.5, .5,.5, 0,blue,1,yellow);
  Draw.DefGrad('rgr10', 2,.5,.5,.2, .5,.5, 0,blue,1,yellow); --
    spreadMethod='repeat',
  Draw.DefGrad('rgr11', 3,.5,.5,.5, .5,.5, 0,blue,1,yellow); --
    spreadMethod='reflect'
  Draw.DefGrad('rgr12', 1,.5,.5,.5, .5,.5, .3,blue,.7,yellow);
  Draw.DefGrad('rgr_text', 1,.5,.5,.8, .5,1, 0,red,.25,yellow,.5,blue,
                   .75,yellow,1,red);
for{i in 1..12} do Draw.Rect('#rgr'&i, 20+200*Trunc((i-1)/4),20+100*(i
                   -1)%4,180,80); end
  --Draw.Text('#rgr_text', 'radialGradient\'s',50,465);
  Draw.Text('#rgr_text', 'radialGradient\'s',50,465);
end
```

Output as follows:
Show linear gradient (xDrawGrad1)

Problem:

Listing 76: The Complete Model in LPL [1]

```lpl
model xDrawGrad1 "Show linear gradient";
Draw.DefFont('Verdana,sans-serif',50);
Draw.DefFill('id',-2,0);
parameter blue:=5; yellow:=6; red:=3; limegreen:=4; green:=13; white :=1;
Draw.DefGrad('lgr1' ,1,0,0,1,0,0, 0,blue,1,yellow);
Draw.DefGrad('lgr2' ,1,0,0,1,0,0, .2,blue,.8,yellow);
Draw.DefGrad('lgr3' ,1,0,0,1,0,0, .4,blue,.6,yellow);
Draw.DefGrad('lgr4' ,1,0,0,1,0,0, .5,blue,.5,yellow);
Draw.DefGrad('lgr5' ,1,0,0,1,0,0, 0,red,1,limegreen);
Draw.DefGrad('lgr6' ,1,0,0,0,1,0, 0,red,1,limegreen);
Draw.DefGrad('lgr7' ,1,0,0,1,1,0, 0,red,1,limegreen);
Draw.DefGrad('lgr8' ,1,0,1,1,0,0, 0,red,1,limegreen);
Draw.DefGrad('lgr9' ,1,0,0,1,0,0, 0,blue,.25,yellow,.5,red,.75,green ,1,blue);
Draw.DefGrad('lgr10',1,0,0,1,0,0, 0,blue,.2,yellow,.8,yellow,1,blue);
Draw.DefGrad('lgr11',1,0,0,1,0,0, 0,blue,.1,yellow,.2,white,.9,blue);
Draw.DefGrad('lgr12',1,0,0,1,1,0, 0,white,.3,yellow,.5,blue,.7,yellow ,1,white);
for{i in 1..12} do Draw.Rect('#lgr'i,20+200*Trunc((i-1)/4),20+100*(i -1)%4,180,80); end
Draw.Text('#lgr9 id','linearGradient\'s',50,465);
end
```

Output as follows:

```
linearGradient's
```

Figure 17: Output
Show figures with gradient (xDrawGrad2)

Problem:

Listing 77: The Complete Model in LPL [1]

```lpl
model xDrawGrad2 "Show figures with gradient"
   // http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html#preface
parameter white:=1; black:=0;
Draw.DefGrad('rGr',-2,-2,-2,-2,-2,-2, 0,white,1,black);
Draw.Path('M',100,200,200,200,150,100); Draw.Path('#rGr','Z',-2,0,2);
Draw.DefGrad('lGr',-2,-2,-2,-2,-2,-2, 0,white,1,black);
Draw.Path('M',210,200,310,200,260,100); Draw.Path('#lGr','Z',-2,0,2);
Draw.DefGrad('rid',1,-2,-2,.65,.05,.05, 0,Rgb(0,238,0), 1,Rgb(0,102,0));
Draw.Rect('#rid',340,100,100,100,-2,Rgb( 0,80,0),3,1,1,10,10);
end
```

Output as follows:

![Figure 18: Output](image)

Figure 18: Output
Check Points inside a Polygon (\texttt{xDrawInside})

**Problem:** The \texttt{Geom.Inside(X,Y,kxa,ky\alpha)} checks whether a point \((X,Y)\) is inside a (simple) polygon consisting of \(k\) points \((x,y,\alpha)\).

**Listing 78:** The Complete Model in LPL [1]

```lpl
model xDrawInside "Check Points inside a Polygon";
 set i:=1..1000 "generate 1000 points (X,Y)";
   k:=1..8;
 parameter PI:=3.14159;
  X{i}:=Rnd(0,1); Y{i}:=Rnd(0,1);
 xa{k}:=.5+Sqrt(1/5)*Sin(PI/#k+2*PI*(k-1)/#k);
 ya{k}:=.5+Sqrt(1/5)*Cos(PI/#k+2*PI*(k-1)/#k);
 Draw.Scale(100,100);
   / \ draw the polygon
  {k} Draw.Path('B',xa,ya); Draw.Path('z',1,0);
   // draw only the points not inside the polygon
 {i|~Geom.Inside(X,Y,{k}xa,{k}ya)} Draw.Circle(X,Y,.005);
end
```

Output as follows:

![Figure 19: Output](image)

Output as follows:
79 Check Two Segments intersect (xDrawIntersect)

Problem:

Listing 79: The Complete Model in LPL [1]

```lpl
model xDrawIntersect "Check Two Segments intersect";
   set i:=1..10 "generate 100 points (X,Y)";
      k:=1..8;
   parameter PI:=3.14159;
      X(i):=Rnd(0,10); Y(i):=Rnd(0,10);
      xa(k):=.5+sqrt(1/5)*sin(PI/#k+2*PI*(k-1)/#k);
      ya(k):=.5+sqrt(1/5)*cos(PI/#k+2*PI*(k-1)/#k);
   Draw.Scale(50,50);
   {i} Draw.Circle(i&'',X,Y,.2,1,0);
   parameter a:=Geom.Intersect(X,Y,8,4,1,2);
   parameter b:=Geom.Intersect(X,Y,5,2,1,10);
end
```

Output as follows:

![Figure 20: Output](image)

Figure 20: Output
80 Drawing Lines (xDrawLine)

Problem:

Listing 80: The Complete Model in LPL [1]

```lpl
model xDrawLine "Drawing Lines";
−− Draw.Scale(3, 3);
−− Draw.Scale(3, −3);
Draw.Line('line', 0, 0, 200, 200);
Draw.CLine('curve', 0, 20, 180, 200, -40);
/* Draw.CArrow('another curve', 0, 40, 160, 200, -40, 0, 3);
Draw.Line(20, 0, 200, 180, 4, 5);
Draw.Line('text follows line', 40, 0, 200, 160, 4, 5, -2, -3);
Draw.DefFont('curs', 'cursive', 30, 3, 2);
Draw.Line('#curs', 'red', 60, 0, 200, 140, 2, 2, -2, 0, 2);
Draw.DefLine('dash', 8, 3, 0);
Draw.Line('#dash', 80, 0, 200, 120);
Draw.Circle(0, 0, 2, 5);
Draw.Text('(0, 0)', 2, 0);
Draw.Circle(200, 200, 2);
*/
end
```

Output as follows:

![Output](image-url)

Figure 21: Output
81 Draw a path (xDrawPath)

Problem:

Listing 81: The Complete Model in LPL [1]

```
model xDrawPath "Draw a path";
  parameter white:=1; grey:=2;
  Draw.DefGrad('rGr',-2,-2,-2,-2,.7,.7, 0,white,1, grey);
  // Draw.Deffill('path',2,2);
  Draw.Rect(0,0,320,320,0,-1);
  Draw.Path('M',10,10);
  Draw.Path('V',20); Draw.Path('H',20);
  Draw.Path('C',20,20,100,10,230,230);
  Draw.Path('S',20,150,100,10);
  Draw.Path('#rGr','Z');
  Draw.Circle(240,280,5,1,0,3);
end
```

Output as follows:

Figure 22: Output
82 Draw a pattern (xDrawPattern)

Problem:

Listing 82: The Complete Model in LPL [1]

```lpl
model xDrawPattern "Draw a pattern";
Draw.Scale(3,3);
set i,j := [1..200];
integer parameter c{i,j} := (i^2+2*j)%32+128;
    --integer parameter c{i,j} := ((i/10)^2+2*j)%32+64;
    --integer parameter c{i,j} := (sqrt(10*i*j))%32+128;
{i,j} Draw.Rect(i,j,1,1,c);
end
```

Output as follows:

![Output](image-url)

Figure 23: Output
83 Load Picture from File (**xDrawPict**)

**Problem:**

Listing 83: The Complete Model in LPL [1]

```plaintext
model xDrawPict "Load Picture from File";
  Draw.Scale(1,1,0,0,700,500);
  Draw.Picture('xDrawPict.jpg');
end
```

Output as follows:

![Figure 24: Output](image)

Another text

Here is text

A Spline curve
84 Some drawing functions (xDrawPict1)

Problem:

Listing 84: The Complete Model in LPL [1]

```lpl
model xDrawPict1 "Some drawing functions"
Draw.Picture('xDrawPict1.jpg',0,0,700,500);  --load and draw a background
Draw.Rect(1,1,100,200,3);  --draw a rectangle
Draw.Line(200,200,300,300,5,20);  --draw an line
Draw.Ellipse(100,200,100,150,7,10,10);  --draw an ellipse
Draw.DefFont('Tahoma',25);  --sets the font and its height
Draw.Text('Here is text',1,1,12);  --write a text to the picture
Draw.Text('Another text',1,300,14,60,90);  --write a text to the picture
Draw.Text('A Spline curve',330,130,12,25,-32);  --write a text to the picture
Draw.CArrow(110,280,480,265,400);  --draw a spline curve
end
```

Output as follows:

```
Here is text
Another text
A Spline curve
```

Figure 25: Output
**Problem:** Look at the graph in Figure 26. Can you see the error?

![Figure 26: 64 ≠ 65](image)

**Modeling Steps**

It is very easy to see that the red and green “triangles” on the right figure are not really triangles (if you are not convinced then set \(n=5\) in the model and run again). A quick calculation also shows that they *cannot* be triangles. The proportion of the height and the length of these triangles is \(\frac{3}{8}\). At the position 5 at the length, the height of this “triangle” is 2. But we know that \(\frac{3}{8} \neq \frac{2}{5}\). Hence, these forms are *not* triangles.

These two graphs are intrinsically linked to the Fibonacci numbers, for example, within the graph the three numbers 5, 8, and 13 are the consecutive Fibonacci numbers \(F_5\), \(F_6\), and \(F_7\). These numbers are defined as follows:

\[
F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad \text{with } n \geq 2
\]

The Fibonacci numbers have many interesting properties, one of them is

\[
F_{n+1} \cdot F_{n-1} - F_n^2 = (-1)^n, \quad \text{with } n \geq 1
\]

**Proof:**

1. The property is true for \(n = 1\), since \(1 \cdot 0 - 1^2 = (-1)^1\).
2. Supposing the property is valid for \(n\), then it is valid for \(n+1\), namely we substitute \(F_{n-1}\) with \(F_{n+1} - F_n\) in the property and we get:

\[
F_{n+1}^2 - F_{n+1}F_n - F_n^2 = (-1)^n
\]

By multiplying with \(-1\) and transforming it we finally get:

\[
F_{n+2} \cdot F_n - F_{n+1}^2 = (-1)^{n+1}
\]

That proves the property.
3. In particular, for \( n = 6 \) we have: \( 13 \cdot 5 - 8^2 = (-1)^6 \) (see Figure 26).

An LPL model that draws this puzzle is given below. Open it and try different values for \( n \) (line 4 in the code).

Listing 85: The Complete Model in LPL [1]

```
model xDrawPuzzle "Puzzle 64=65"

set k:=1..20;
parameter F{k}:=if(k=1,1,k=2,1,F[k-1]+F[k-2]); //Fibonacci numbers
n:=7; -- for different sizes: try n:=5,6,7 and 8
x:=2; y:=2;
set i:=1..F[n+1]+10;
Draw.Scale(25,25);
Draw.DefFont('Verdana',40);
{i}Draw.Line(1,i,#i,i,2);
{i}Draw.Line(i,1,i,#i,2);
Draw.Path('M',x,y,x+F[n-1],y,x+F[n-2],y+F[n-1],x,y+F[n-1]);
Draw.Path('Z',2,0,3,.5);
Draw.Path('M',x,y+F[n-1],x+F[n],y+F[n-1],x+F[n],y+F[n]);
Draw.Path('Z',5,0,3,.5);
Draw.Path('M',x,y+F[n-1],x+F[n],y+F[n],x,y+F[n]);
Draw.Path('Z',3,0,3,.5);
Draw.Path('M',x,y+F[n-1],x+F[n],y+F[n],x,y+F[n]);
Draw.Path('Z',4,0,3,.5);
Draw.Text(F[n]&' x '&F[n]&' = '&F[n]^2,x+F[n]+2,4);
```

This puzzle shows an important aspect in the modeling process: How can we discover generalizations of a result! Try to construct the puzzle for \( n = 7 \): we get

\[
21 \cdot 8 - 13^2 = (-1)^7 \quad (168 = 169 - 1)
\]

The corresponding figure can be constructed and is shown in Figure 27.

In the puzzle with \( n = 5 \) (Figure 28), we see immediately, that the two forms are not congruent.
Figure 27: \(169 = 168\)

\[8 \times 21 = 168\]

\[13 \times 13 = 169\]

Figure 28: \(25 = 24\)

\[3 \times 8 = 24\]

\[5 \times 5 = 25\]
86   Draw rectangles (xDrawRect)

Problem:

Listing 86: The Complete Model in LPL [1]

```lpl
model xDrawRect "Draw rectangles";
  −−Draw.Scale(3,3);
  Draw.Scale(3,-3);
  Draw.Rect('a',100,100,30,40,2,3,5);
  Draw.Rect('rect',100,70,20,20,1,0);
  Draw.Rect('12.0',60,0,30,20,2,3,4,0.2,0.2);
  Draw.Rect('14.0',95,0,40,40,2,3,4,-2,-2,10,10);
  Draw.Rect(0,0,150,150,-1,3,2); // the large rect
  parameter Pi:=3.14159; x1:=100; y1:=45;
      an:=Arctan(y1/x1)*180/Pi;
      r:=Sqrt(x1^2+y1^2);
      x2:=r*Cos((an+50)*Pi/180); y2:=r*Sin((an+50)*Pi/180);
  Draw.Text('(x1,y1)',x1+3,y1+2);
  Draw.Text('(x2,y2)',x2-21,y2-3);
  Draw.DefTrans('tr','t',x1,y1);
  Draw.DefTrans('ro','r',50);
  Draw.Rect('#ro tr','hu',0,0,20,30,1,0);
  Draw.Arc('α=50º',0,0,r,an,an+50,3,10);
  Draw.Circle(x1,y1,2);
  Draw.Line(0,0,x1,y1);
  Draw.Circle(x2,y2,2);
  Draw.Line(0,0,x2,y2);
  parameter x:=50; y:=60;
  Draw.DefTrans('ro1','r',50,x,y);
  Draw.Rect('#ro1','hi',x,y,20,30,-1,0);
  Draw.Circle(x,y,2);
  Draw.Text('(x,y)',x+2,y-2);
end
```

Output as follows:


Figure 29: Output
87 Draw.Scale Function (xDrawScale)

**Problem**: The first Scale instruction scales the graph automatically, the second does manually: the size of the graphic is (240,240) and the left/top of the graphic is (-120,-120) and (2,-2) are the scaling factors.

Listing 87: The Complete Model in LPL [1]

```
model xDrawScale "Draw.Scale Function";
    −−Draw.Scale(2,2);
    Draw.Scale(2,-2,-120,-120,240,240);
    Draw.Rect(-100,-100,200,200);
    Draw.Circle(-100,-100,2,3);
    Draw.Circle(100,100,2,4);
    Draw.Circle(-100,100,2,3);
    Draw.Circle(100,-100,2,4);
end
```

Output as follows:

![Output](image)

Figure 30: Output
88 Drawing Text (xDrawText)

Problem: Text fonts in different shapes and sizes.

Listing 88: The Complete Model in LPL [1]

```lpl
model xDrawText "Drawing Text";
  −− Draw . Scale ( 3 , 3 ) ;
  Draw . Scale ( 3 , 3 , 0 , - 10 , 200 , 150 ) ;
  parameter FontSize := 30 ;
  −− Draw . DefFont ( ' serif ' , FontSize ) ; // default
  Draw . DefFont ( ' serif ' , ' serif ' , FontSize ) ;
  Draw . DefFont ( ' sans ' , ' sans-serif ' , FontSize ) ;
  Draw . DefFont ( ' curs ' , ' cursive ' , FontSize ) ;
  Draw . DefFont ( ' fant ' , ' fantasy ' , FontSize ) ;
  Draw . DefFont ( ' mono ' , ' monospace ' , FontSize ) ;
  Draw . Text ( ' Hello World! default ' , 20 , 0 ) ;
  Draw . Text ( ' # seri ' , ' Hello World! serif ' , 20 , 20 ) ;
  Draw . Text ( ' # sans ' , ' Hello World! sans-serif ' , 20 , 40 ) ;
  Draw . Text ( ' # curs ' , ' Hello World! cursive ' , 20 , 60 ) ;
  Draw . Text ( ' # fant ' , ' Hello World! fantasy ' , 20 , 80 ) ;
  Draw . Text ( ' # mono ' , ' Hello World! monospace ' , 20 , 100 ) ;
  Draw . DefFill ( ' fill ' , 3 , 4 , 3 , 0 . 2 , 0 . 5 ) ;
  Draw . Text ( ' # fant fill ' , ' Hello World!!! ' , 20 , 130 , 100 ) ;
  Draw . Circle ( 0 , 0 , 1 ) ;
  Draw . Text ( ' ( 0 , 0 ) ' , 2 , 0 ) ;
end
```

Hello World! serif
Hello World! sans-serif
Hello World! cursive
Hello World! fantasy
Hello World! monospace

Figure 31: Output
89 Drawing a TextPath (xDrawTextPath)

Problem:

Listing 89: The Complete Model in LPL [1]

```lisp
model xDrawTextPath "Drawing a TextPath";
   Draw.Scale(2,2,0,0,150,200);
   — this defines only a path with id 'aa'
   Draw.Path('aa','M',50,50);
   Draw.Path('aa','A',30,30,0,0,1,130,80);
   Draw.Path('aa','L',130,180);
   — this uses the path for the text
   Draw.Text('#aa','Text on a round corner, and more text. hello ',0,0,28);
end
```

Output as follows:

![Output](image)

Figure 32: Output
Draw a Text on a Path ([xDrawTPath](#))

**Problem:**

Listing 90: The Complete Model in LPL [1]

```lpl
model xDrawTPath "Draw a Text on a Path";
Draw.Scale(1,1,-5,145,360,210);
Draw.Rect(0,150,350,200,1,0);
Draw.Path('mytext','M 20,255 L 100,255 A 70 100 0 0 0 180,255 A 70
100 0 0 1 260,255 L 370,255 y',-1,3,1);
Draw.Text('#mytext','this is a longer text on a path with downs and ups',100,100);
end
```

Output as follows:

![Output](image)

Figure 33: Output
91  **Show transformations (xDrawTrans1)**

**Problem:**

Listing 91: The Complete Model in LPL [1]

```plaintext
model xDrawTrans1 "Show transformations";
Draw.Scale(2,2,0,0,130,100);
−−Draw.Scale(2,2);//does not a good job!
for{i in 1..10} do
  Draw.DefTrans('id'&i,'t',10+3*i,10);
  Draw.DefTrans('id'&i,'r',10*i,40,40);
  Draw.Rect('#id'&i,10,10,50,50,i+3,i+2,3,2,2,.1,.1);
end;
−−Draw.DefTrans('id1','r',45,130,30);
−−Draw.Rect('#id1',120,20,50,50);
−−Draw.Rect(100,100,90,90);
end
```

Output as follows:

![Figure 34: Output](image-url)
92 Draw Penrose Triangle (xDrawPenrose)

Problem:

Listing 92: The Complete Model in LPL [1]

```
model xDrawPenrose "Draw Penrose Triangle";
parameter a:=300; b:=40; c:=Sqrt(1/3)*b;
    d:= a-3*c; e:=Sqrt(3/4)*(d+c)-2*b; f:=1/2*a-3*c;
Draw.Scale(1,1,0,10,500,500);
Draw.Rect(0,0,700,700);
Draw.DefGrad('nr1',3,0,0,1,1,-2,0,Rgb(2,30,40),1,1);
Draw.DefTrans('trans1','r',120,100+f+2*c,400-e-4*b);
Draw.DefTrans('trans2','r',240,100+a+c,400-b);
Draw.Path('M',100,400); Draw.Path('l',a,0); Draw.Path('l',c,-b);
    Draw.Path('l',-c-d,0); Draw.Path('l',f,-e); Draw.Path('l',-c,-b);
    Draw.Path('#nr1','Z');
Draw.Path('M',100+f+2*c,400-e-4*b); Draw.Path('l',a,0); Draw.Path('l',c,-b);
    Draw.Path('l',-c-d,0); Draw.Path('l',f,-e); Draw.Path('l',-c,-b);
    Draw.Path('#nr1 trans1','Z');
Draw.Path('M',100+a+c,400-b); Draw.Path('l',a,0); Draw.Path('l',c,-b);
    Draw.Path('l',-c-d,0); Draw.Path('l',f,-e); Draw.Path('l',-c,-b);
    Draw.Path('#nr1 trans2','Z');
end
```

Output as follows:

![Figure 35: Output](image-url)
Draw Sierpinski Triangles (*xDrawxSierpinski*)

Problem:

Listing 93: The Complete Model in LPL [1]

```
model xDrawSierpinski "Draw Sierpinski Triangles"
    parameter l:=400; t:= 10; c; f:=3^Ceil(Log(l/t)/Log(2));
    set i,j:=1..f;
    parameter x{i}; y{i}; xx{j}; yy{j};
    x[1] := 1; y[1] := 1;
    while l>t do
        l := l/2; c := 1;
        {i|x}(xx[c]:=x, yy[c]:=y, xx[c+1]:=x+l, yy[c+1]:=y,
            xx[c+2]:=x+l/2, yy[c+2]:=y+Sqrt(3/4*l^2), c:=c+3);
        {j}(x[j]:=xx, y[j]:=yy);
    end
    Draw.DefFill('pa',1,0,-2,0.5);
    {i|x}(Draw.Path('M',x,y,x+l,y,x+l/2,y+Sqrt(3/4*l^2),x,y),Draw.Path('#pa','Z'));
end
```

Output as follows:

Figure 36: Output
Draw a colored spiral (xDrawSpiral)

Problem:

Listing 94: The Complete Model in LPL [1]

```lpl
model xDrawSpiral "Draw a colored spiral";
  set r := [1..100];
  parameter f:=14; a:= 4; c; d; l; size:=400; rad := a/180*3.14159265;
  Draw.DefGrad('grad0',3,0,0,1,1,-2,0,5,1,6);
  Draw.Rect('#grad0',0,0,size,size);
  l:=size;
  {r}(Draw.DefTrans('rot','r',a,size/2,size/2),
    Draw.DefGrad('grad'r,3,0,0,1,1,-2,0,5,1,6),
    c:=Arctan(rad)*l/(1+Arctan(rad)), d:= l-c, l:=Sqrt(c^2+d^2),
    Draw.Rect('#rot grad'r,(size-l)/2,(size-l)/2,1,1));
end
```

Output as follows:

![Figure 37: Output](image-url)
Problem:

Listing 95: The Complete Model in LPL [1]

```lpl
model xDrawArcPath "Drawing Library I";
    Draw.Scale(3,3);
    Draw.Rect(0,0,155,155,2);
    Draw.Line(5,5,5,150,0);
    Draw.Line(5,5,150,5,0);
    Draw.Line(5,75,75,5,0);
    Draw.Line('ff',5,150,50,5,0);
    Draw.Text('A',1,4);
    Draw.Text('B',1,75);
    Draw.Text('C',40,42);
    Draw.Text('D',51,4);
    Draw.Text('y',1,150);
    Draw.Text('x',145,4);
    Draw.Circle(38.5,41.5,1.5);
    Draw.Text('6x+2y = 300',27,100,-75,16);
    Draw.Text('5x+5y = 350',50,35,15,0,0,1,-2,-2,-45,-45); // x,y,h,c1,c2,w,o1,o2,a,r
end
```

Output as follows:

Figure 38: Output
References