LPL Tutorial

Tony Hürlimann
Department of Informatics
University of Fribourg

October 31, 2008

Abstract

Several small LPL model examples guides you through certain capacities of the modelling system LPL. It is only a first overview, the complete language specification is found in the reference manual *manual.pdf*.

⟦This PDF-document (*tutor.pdf*) was generated automatically from the Latex file *tutor.tex* and the LPL source files using LPL's own documentation tool. How this is done is explained in the manual (Chap 10.5).⟧

Contents

1   LPL Tutorial

The following examples start with a simple small production model. Successively, more features are added to the model to illustrate the syntax of the language LPL.

This text is an automatically generated text using the documentation facility of LPL, and illustrates itself an important feature of LPL. Going through the examples together with loading and running them using *lplw.exe* gives you a first overview of the capacity of the modelling 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 modelling language. Enjoy it!

2   A Simple Model (tutor01)

⟦The problem is briefly explained in this part.⟧ A firm produces two type of   **Problem**
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 Marie and 8 hours for Jules,

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

The profit 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 profit?

Modeling Steps

⟦This part explains how the problem can be translated into a mathematical model.⟧

1. We 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: $5x + 5y \leq 350$, the component capacity per week.

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: $x \geq 20, x \geq 15$.

6. Maximizing the profit means to maximize: $300x + 200y$.

The problem to solve is then as follows:

$$
\begin{array}{ll}
\text{max} & 300x + 200y \\
\text{subject to} & 5x + 5y \leq 350 \\
& 4x + 8y \leq 300 \\
& 6x + 2y \leq 300 \\
& x \geq 20 \quad y \geq 15
\end{array}
$$

In a graphical way, the model represents the solution space as shown in **Comments** 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.
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR01 "A Simple Model";
  VARIABLE  x; y;
  MAXIMIZE profit: 300*x + 200*y;
  CONSTRAINT
    Components: 5*x + 5*y  <=  350;
    Mounting:   4*x + 8*y  <=  480;
    Testing:    6*x + 2*y  <=  300;
    Order1:       x        >=   20;
    Order2:             y  >=   15;
  WRITE x,y;
END
```
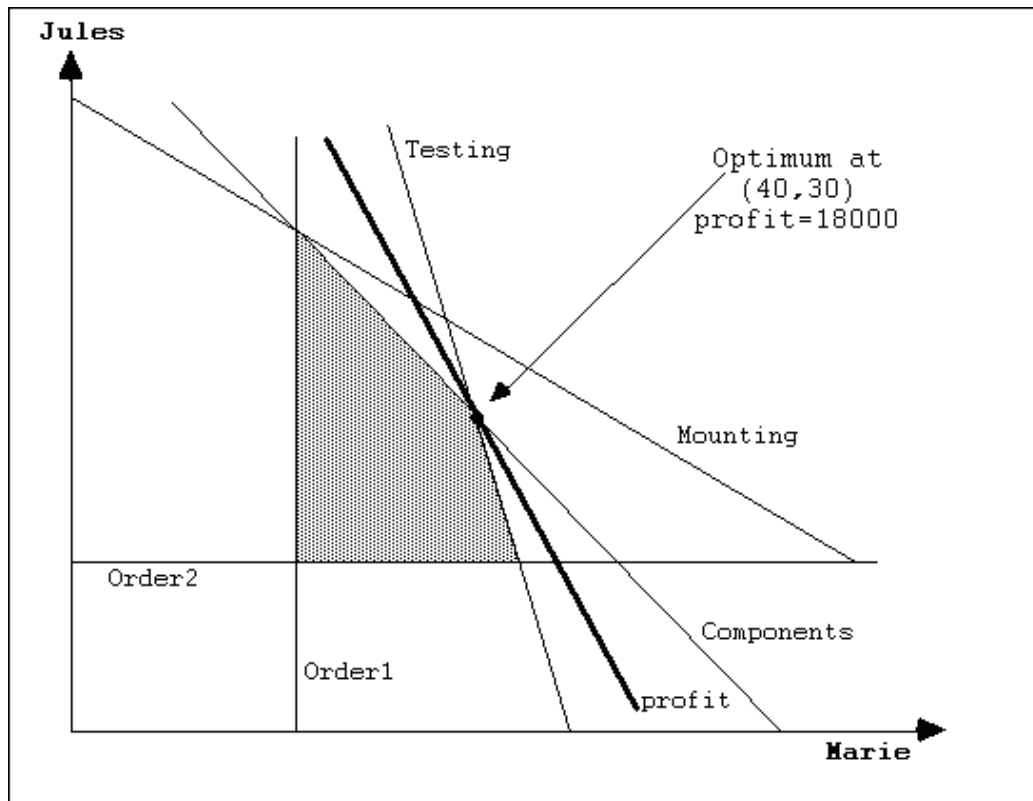
Figure 1: The Feasible Space

LPL Modeling Steps

⟦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 is very close to the usual mathematical notation.

1. Each model coded in LPL begins with a keyword `MODEL` and ends with `END`. (Keywords can be in lower or uppercase.)

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

3. Each entity begins with an 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

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

6. The objective function, called `profit`, begins with `MAXIMIZE`.

7. Finally, we want to write the solution with `WRITE`.

⟦This part of the documentation is intended to gives a short comment about **Solution** 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 `profit`, and hence it is no longer maximized.

Question 1 *(Answer to 1)*

⟦*To animate the reader to learn actively, some questions are proposed. Answers are given below.*⟧

1. *Verify the solution of this model by running it with* `lplw.exe`.

2. *What happens if we extend the* `Mounting` *capacity to* 500*?*

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

Answer 1 *(Question of 1)*

1. *Run* `lplw.exe`. *Then 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.*

2. *Nothing, 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, changes nothing. To calulate it, change* 480 *to* 500 *in the model and run it again. Then look at the values of the variables again.*

3. *Change the constraint* `Order2` *to* `y >= 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 profit also drops to* 16500*. This can be seen by clicking the blue node* `profit` *while 'TABLE' is the active tab.*

6

3   Names and Comments (tutor02)

This is exactly the same model as `tutor01.lpl`. Comments are added.   **Problem**
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR02 "Names and Comments";

     (* First the variable are declared *)
  VARIABLE Marie ALIAS x "Number of robots Marie";
    Jules ALIAS y        "Number of robots Jules";

     (*  Now follows the constraints *)
  CONSTRAINT
    Components: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 is maximized *)
  MAXIMIZE profit: 300*x + 200*y;

     -- finally we print the results
  WRITE profit, 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`. A second name is introduced by the keyword
   `ALIAS`.

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

3. Short one-line comments begin with a double-dash (`--`). 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 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 2 *(Answer to 2)*

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

2. Outcomment the line of `Jules ALIAS...`*. What happens?*

3. What happens if we change the name `Marie` *to* `Mary`*?*

Answer 2 *(Question of 2)*

1. The constraint `Testing` *is outcommented 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.

4 Using Indices (tutor03)

⟦This model now introduces the very fundamental concept of *index*.⟧
Suppose now we have 10 different type of robots (not just 2 like in the previous models. A way to deal with this is to introduce 10 different variables. However there is a more economical way: Using indexes. In mathematical notation we would introduce this as:

$$x_i, \quad \text{with } i \in I = \{1 \ldots 10\}$$

in LPL syntax we might use exactly this same notation and we can write:

```
SET I := /1:10/;
VARIABLE x{i in I};
```

However, in LPL normally there is not need to make a difference between the indexname `i` and the setname `I` if no confusion arises. Hence we can just write:

```
SET i := /1:10/;
VARIABLE x{i};
```

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR03 "Using Indices";

SET  i := / 1:10 / "A set with 10 elements";
VARIABLE  x{i} "The number of different type of robots";
PARAMETER HC{i} := [ 5 5 4 5 6 5 7 8 4 7 ] "Component time";
     HM{i} := [ 4 8 5 6 4 8 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 ];

CONSTRAINT
```

```
   Components: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";
MAXIMIZE profit: SUM{i} Price[i]*x[i] "Maximize the profit";

WRITE profit, x, HC, HM, HT "output the data";
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.

2. `VARIABLE` introduces a variable list called `x`, which is indexed over i. This declares the 10 variables: `x[1]` ... `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 `x[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]*x[1] + ... + HC[10]*x[10]
   ```

   is written as: `SUM{i} HC[i]*x[i]`. This is LPL's notation for

   $$\sum_{i\in\{1...10\}} (HC_i \cdot x_i)$$

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

```
Order{i}: x[i] >=  Ordered[i];
```

declares 10 constraints:

```
x[1] >= Ordered[1], x[2] >= Ordered[2], ...
```

In mathematical notation, we would write this as follows:

$$x_i \geq Ordered_i, \quad i \in \{1 \ldots 10\}$$

Note again, that no difference is made in LPL's notation between the index-name and the set-name.

7. A `WRITE` instruction can be used to write the results to the `NOM-file`. In this case, tables are written to the file `tutor03.nom`.

Question 3 *(Answer to 3)*

1. *Replace* `1:10` *by* `1:11` *in the set definition of* `i`. *What happens?*

2. *Change* `Price{i}` *to* `Price{I}` *in the price declaration. What happens?*

Answer 3 *(Question of 3)*

1. *An error occurs because the data list only contain 10 (not 11!) data elements. Adding a numerical data to each of the data list then allows again to run the model.*

2. *An error occurs, because* `I` *is not declared. Note, LPL ise case-sensitive, hence,* `i` *and* `I` *are two different names (keywords however can be in lower or upper-case.)*

5 Using Units (tutor04)

This is the same model as `tutor03.lpl`. We added some units of measure- **Problem**
ments. Expressions using the units of measurement are checked for correct
expressions. Hence, if one adds something measured in meter and something
in kilogram then LPL can detect the error and reports it.
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR04 "Using Units";

UNIT
  piece ALIAS p;          -- number of pieces
  dollar ALIAS d;         -- money unit
  hour ALIAS h;           -- time unit
  d100 := 100*d;          -- another compatible money unit
  day  := 8*h;            -- another time unit
  week ALIAS w := 5*day;  -- still another time unit
  DailyIncome := d/day;   -- a compound unit

SET  i := /1:10/;         -- a set of 10 elements

VARIABLE  x{i} UNIT [p/w];  -- number of robots per week

PARAMETER HC{i} UNIT [h/p] := [ 5 5 4 5 6 5 7 8 4 7 ];
          HM{i} UNIT [h/p] := [ 4 8 5 6 4 8 7 6 5 3 ];
          HT{i} UNIT [h/p] := [ 6 2 4 6 3 4 5 2 5 3 ];
  Ordered{i} UNIT [p/day]  := [160 120 56 48 40 64 72 64 56 40];
  Price{i} UNIT [d100/p]   := [ 3 2 1 0.5 0.5 1 2 1 4 2 ];

CONSTRAINT
  Components UNIT [h/w]: SUM{i} HC[i]*x[i] <= 140000[h/w];
  Mounting UNIT [h/w]:   SUM{i} HM[i]*x[i] <= 38400[h/day];
  Testing UNIT [h/w]:    SUM{i} HT[i]*x[i] <= 120000[h/w];
  Order{i} UNIT [p/w]:   x[i]              >= Ordered[i];
MAXIMIZE profit UNIT [d/w]: SUM{i} Price[i]*x[i];

WRITE profit;
WRITE UNIT [d/h] : profit;
WRITE UNIT [DailyIncome] : profit;
WRITE HC, Price;
WRITE x;
```

```
    WRITE{i} UNIT [p/day] : x;
    END
```

LPL Modeling Steps

A new declaration is introduced: the UNIT statement.

1. Units can be declared and defined in a Unit entity initiated by `UNIT`.

2. Basic units like `piece`, `dollar`, or `hour` are just declared by their names. ALIAS can be used to give them a second name.

3. Derived units like `day` or `DailyIncome` are defined in terms of other units. So, for example, a week *is* 5 days: `w := 5*d;`

4. A variable, data, or constraint can be declared together with a unit by adding the reserved word `UNIT` and a unit expression within brackets. For example, the variables `x` are measured in 'pieces per week', hence in `[p/w]`.

5. Numbers in expressions must be extended with a unit expression within `[` and `]` to satisfy the commensurability of expressions.

6. `WRITE` statements can be extended with a compatible units, the transformation are done automatically.

Question 4 *(Answer to 4)*

   1. *Change* `140000[h/w]` *to* `100000[d]`*, then run the model. What do you see?*

   2. *Change the identifier* `hour` *to* `Hour`*. What effect has this on the next run?*

Answer 4 *(Question of 4)*

   1. *An error occurs, because the unit of the right hand side of the* `Conponents` *constraint does not correspond to the declared one on the left side.*

   2. *No effect! Since* `Hour` *is not used anywhere else in the model.*

6 Data Tables (tutor05)

This is the same model as `tutor03.lpl`. The data are represented differently **Problem** (more like in a 3-norm database form).

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR05 "Data Tables";

-- a set i together with five data vectors.
SET  i :=/ | HC  HM  HT  Ordered  Price |
    Robot1   5   4   6     20      300
    Robot2   5   8   2     15      200
    Robot3   4   5   4      7      100
    Robot4   5   6   6      6       50
    Robot5   6   4   3      5       50
    Robot6   5   8   4      8      100
    Robot7   7   7   5      9      200
    Robot8   8   6   2      8      100
    Robot9   4   5   5      7      400
    Robot10  7   3   3      5      200  /;

VARIABLE  Robots{i};
CONSTRAINT
  Components: SUM{i} HC*Robots <= 3500;
  Mounting:   SUM{i} HM*Robots <= 4800;
  Testing:    SUM{i} HT*Robots <= 3000;
  Order{i}:   Robots  >=  Ordered;
MAXIMIZE profit: SUM{i} Price*Robots;
WRITE profit;
WRITE '
Robots   Rob-Ord   Price   Robots  Tot.Hours
%7s    %3d       %3d       %3d      %3d\n' :
  ROW{i}(i, Robots-Ordered, Price, Robots, HC+HM+HT);
END
```

LPL Modeling Steps

All data are in a unique "table" in text-format. The format of this table is somewhat special and I guess that it is not used very often. But now you know that it exists! More often, the modeller will store the data in databases.

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

2. The data which are indexed over the same sets can be compiled in a convenient table format. This format – surrounded by `/` is called the *format B*.

3. The indices in expressions can be dropped, since LPL already knows that `Robots`, for example, has been defined over the index-set `i`. Hence the two following lines are equivalent for LPL:

    ```
    SUM{i} HC * Robots <= 3500
    SUM{i} HC[i] * Robots[i] <= 3500
    ```

4. The WRITE may also be used to write a list of expressions defined by a *format mask*. This can write quite complex output format. In this case, a list of the five following expressions

    ```
    i, Robots-Ordered, Price, Robots, HC+HM+HT
    ```

    is written line by line over all `i`. The mask is given within apostrophes (`'...'`) and the format of the single data is indicated by `%` followed by a letter. The syntax is the same as in Java or C. So, `%7s` means to fill the first parameter (`i`) with 7 chars, `%3d` means to fill in an integer with 3 positions.

Question 5 *(Answer to 5)*

1. *Modify* `%3d` *in the WRITE just below* `Robots` *to* `%6.2f` *and run again. What's the difference?*

2. *The previous question shows that some number of Robots have a fractional number. This is non-sense. The number of robots must always be an integer number. Change the the model correspondingly.*

3. *How much is the profit, if no robot has been ordered in advance?*

Answer 5 *(Question of 5)*

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

2. *All you need to do is to modify the line* `VARIABLE  Robots{i};` *with* `INTEGER VARIABLE  Robots{i};` *.*

   However, how LPL's internal solver cannot not solve this problem anymore. You need an integer solver – the free GNU GLPK solver, for example. To install it follow the instructions given in the `lplcfg.lpl` *file (look for "GLPK solver" in the file).*

3. *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.*

7 Data Tables I (tutor06)

We introduce a second index-set j. We now suppose that there are not 3 but 8 production steps. They are collected in the new index-set j. **Problem**
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR06 "Data Tables I";
                                    -- data indexed over i
SET    i  := / | Ordered Price |

      Robot1  20   300   Robot2   15   200
      Robot3   7   100   Robot4    6    50
      Robot5   5    50   Robot6    8   100
      Robot7   9   200   Robot8    8   100
      Robot9   7   400   Robot10   5   200 /;


                                    -- data indexed over j
       j  := / | Capacity |
      Step1  3500   Step2  4800   Step3  3000   Step4  3400
      Step5  3000   Step6  3200   Step7  4000   Step8  2500 /;


                                    -- data indexed over {i,j}
PARAMETER Hours{i,j} := /
     : Step1  Step2  Step3  Step4  Step5  Step6  Step7  Step8:
  Robot1  9      9      9      9      9      9      .      9
  Robot2  5      8      2      .      3      .      .      3
  Robot3  4      5      4      .      5      6      .      5
  Robot4  5      6      6      .      8      .      .      4
  Robot5 10      4      3      .      .      5      .      6
  Robot6  5      8      4      .      .      .      5      1
  Robot7  7      7      5      5      .      3      .      2
  Robot8  8      6      2      4      .      .      .      5
  Robot9  4      5      5      6      4      .      .      .
  Robot10 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   profit: SUM{i} Price[i] * Robots[i];
WRITE profit, Robots;
```

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 using a convenient format (data *format B*).

4. Note that the `Steps{i}` constraint generates 8 different constraints. One can verify this when generating the `EQU-file` (in lplw.exe use menu 'tools/EQU-file').

Question 6 *(Answer to 6)*

1. *Run the model, then click the menu Tools/Create EQU-file.*

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

Answer 6 *(Question of 6)*

1. *An equation listing is generated. It is stored in the* `EQU-file`. *On disk the file is:* `tutor06.equ`.

2. *Run the model then click the tab 'TABLE' and then click the blue cycle 'Steps' at the left part of the application. Then click the radio button 'Du' (for dual values, or reduced prices). Only* `Step1` *and* `Step3` *have a reduced price different from zero. Hence, these two constraints are "stringent". All others have superfluous capacities.*

8 Data Tables II (tutor07)

The same model as `tutor06.lpl`. The data are in another file.
The complete model code in LPL for this model is as follows:

Problem

```
MODEL TUTOR07 "Data Tables II";

  -- the data are defined in file 'tutor.inc'
  (*$I 'tutor.inc' *) -- the file is included here

  VARIABLE  Robots{i};
  CONSTRAINT
    Steps{j}: SUM{i} Hours * Robots <= Capacity;
    Order{i}:   Robots  >=  Ordered;
  MAXIMIZE   profit: SUM{i} Price * Robots;
  WRITE profit, Robots;
  WRITE{i}: Robots.dual;
END
```

The included data file – also specified in LPL syntax – is as follows:

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

  -- data indexed over i
  SET  i  := / | Ordered Price |
      Robot1  20  300   Robot2  15  200
      Robot3   7  100   Robot4   6   50
      Robot5   5   50   Robot6   8  100
      Robot7   9  200   Robot8   8  100
      Robot9   7  400   Robot10  5  200 /;

  -- data indexed over j
      j  := / | Capacity |
      Step1 3500   Step2 4800   Step3 3000   Step4 3400
      Step5 3000   Step6 3200   Step7 4000   Step8 2500 /;

  -- data indexed over {i,j} , a two dimensional table
  PARAMETER Hours{i,j} ALIAS H := /
        : Step1 Step2 Step3 Step4 Step5 Step6 Step7 Step8:
   Robot1    9     9     9     9     9     9     .     9
```

```
Robot2    5    8    2    .    3    .    .    3
Robot3    4    5    4    .    5    6    .    5
Robot4    5    6    6    .    8    .    .    4
Robot5   10    4    3    .    .    5    .    6
Robot6    5    8    4    .    .    .    5    1
Robot7    7    7    5    5    .    3    .    2
Robot8    8    6    2    4    .    .    .    5
Robot9    4    5    5    6    4    .    .    .
Robot10   7    3    3    1    1    1    1    4 /;
```

LPL Modeling Steps

Physical inclusion of files:

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

2. The data are defined in another file (here `tutor.inc`), which is automatically included at parse time.

3. The second write statement prints the dual values of the `Robots` variable. The write 'Runs through' the index `i` and prints the expression `Robots[i].dual` for each `i`.

Question 7 *(Answer to 7)*

1. *Change* `Robots.dual` *with* `Robots.lran` *and execute.*

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

Answer 7 *(Question of 7)*

1. *You'll get the lower sensitivity bound for the variables (see the theory of sensitivity analysis).*

2. *The file* `tutor.inc` *opens in a new tabbed window.*

9   Reading Textfiles (tutor08)

The data are read in from a text-file by an instruction READ. **Problem**
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR08 "Reading Textfiles";

  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 the data here
  READ FROM 'tutor.txt' '%1:Table';
  READ '%1' : ROW{i} (i , Ordered , Price);
  READ '%2' : ROW{j} (j , Capacity);
  READ '%3' : COL{j} j , ROW{i} (i , COL{j} Hours);

  MAXIMIZE profit: SUM{i} Price*Robots;
  WRITE Robots, Hours;
END
```

The data file read by LPL is:

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

Table 1 : reads i, Ordered, and Price   (first block)
    Robot1   20      300
    Robot2   15      200
    Robot3    7      100
    Robot4    6       50
    Robot5    5       50
    Robot6    8      100
    Robot7    9      200
    Robot8    8      100
    Robot9    7      400
```

```
   Robot10   5      200

Table 2  : reads j, and Capacity        (second block)
   Step1  3500
   Step2  4800
   Step3  3000
   Step4  3400
   Step5  3000
   Step6  3200
   Step7  4000
   Step8  2500


Table 3  : reads Hours                   (third block)
        Step1  Step2  Step3  Step4  Step5  Step6  Step7 Step8
   Robot1  9      9      9      9      9      9      .     9
   Robot2  5      8      2      .      3      .      .     3
   Robot3  4      5      4      .      5      6      .     5
   Robot4  5      6      6      .      8      .      .     4
   Robot5 10      4      3      .      .      5      .     6
   Robot6  5      8      4      .      .      .      5     1
   Robot7  7      7      5      5      .      3      .     2
   Robot8  8      6      2      4      .      .      .     5
   Robot9  4      5      5      6      4      .      .     .
   Robot10 7      3      3      1      1      1      1     4
```

LPL Modeling Steps

The data are separated from the model file and can be read by instruction.

1. One can use the READ statement to read an external data-textfile. The data must be in a particular row-, columnwise form.

2. The first READ instruction tells us from which file to read and what the block-delimiter are (here Table). A line in the data file beginning with this string begins a new block in the file. The blocks are numbered beginning with 1. The line containing the block delimiter is not considered for reading. The reading begins on the following line.

3. Subsequent READ statement will then read from this file. For example: READ '%1' ... means to read the first block beginning with the string Table and reading till the next block-delimiter.

4. The expression `ROW{i} (i,Ordered,Price)` means to read three tokens per row (line) and assign the data to the three vectors. The tokens read are separated by a comma within the expression. In the text file they are separated by a blank, a tab or some other special characters.

5. `ROW` and `COL` are two keywords which instruct to read row- and column-wise. So, `COL{j} j` reads on a single line all tokens and assign them to j. Then `ROW{i} (i,COL{j} Hours[i,j])` reads on each row first one token i and then all tokens over j on that line to assign then to `Hours[i,j]`.

Question 8 *(Answer to 8)*

1. Exchange the second and the third `READ` *instruction. What happens?*

2. Outcomment the `MAXIMIZE` *instruction then execute. What happens?*

Answer 8 *(Question of 8)*

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

2. The model is not solved. However, that data are nevertheless read from the textfile. This can be seen when adding an instruction `WRITE 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 and the data of* `Hours` *are listed.*

10 Writing Masks (tutor09)

This is the same model as `tutor08.lpl`. However, now we use masks for **Problem** model reporting using the `WRITE` statement.

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR09 "Writing Masks";

  (*$I 'tutor.inc' *)  -- read the data

  PARAMETER CpH := 5 "Cost per hour";
  --Price{i}:=2*Price;
  VARIABLE Robots{i};
  CONSTRAINT
    Steps{j}: SUM{i} Hours * Robots <= Capacity;
    Order{i}:  Robots  >=  Ordered;
  --MAXIMIZE revenue: SUM{i} Price*Robots;
  MAXIMIZE profit: SUM{i} Price*Robots -SUM{i,j} CpH*Hours*Robots;

  WRITE '
    OUTPUT of the TUTOR09 model
    ---------------------------
  type of    number of    already    Price    Cost
   robot       robot       ordered    /unit    /unit
-----------------------------------------------------
   %7s    %4d        %3d        %3d      %3d   %15s
-----------------------------------------------------

     Total of revenue :        %9.2f
     Total of costs   :        %9.2f
                               ---------
     Total of profit  :        %9.2f\n'

 : ROW{i} (i , Robots , Ordered , Price , SUM{j} CpH*Hours ,
           IF(Price<SUM{j}CpH*Hours,'loosing money','ok') ) ,
   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. The mask is entered between ' ... ' and instructs LPL how to format the output. Format specifiers begin with a `%`.

2. The `WRITE` expression instructs LPL how to fill this mask. `ROW{i}` means to write the following expressions as many times as i has elements each on a horizontal line.

3. The `ROW` fills 6 mask elements with data: the first format specifier (`%7s`) is filled by `i`, the second `%4d` is filled by `Robots`, the third `%3d` by `Ordered`, the fourth by `Price` the fifth by `SUM{j} HourCost*Hours[i,j]` (that is, by $\sum_j HoursCost \cdot Hours_{i,j}$), and the last by `loosing money`, whenever the `Price` is smaller than the fifth element. (`IF(a,b,c)` means "return `b` if `a` is true, else return `c`").

Question 9 *(Answer to 9)*

1. *Double the* `Price` *vector and see what happens.*

2. *The model maximizes the total revenue:* `SUM{i} Price*Robots`. *Hence the profit (revenue minus cost) will be* 141945. *How will the profit be if we maximize the profit.*

Answer 9 *(Question of 9)*

1. *To double the prices, we only need 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 loose money on `Robot4` *and* `Robot5`.

2. *To maximize the profit, one only needs to add the cost to the maximizing function the maximizing function then is as follows:*

   ```
   MAXIMIZE profit: SUM{i} Price*Robots -SUM{i,j} CpH*Hours*Robots;
   ```

   Running the model then displays a profit of 146208.

11 Sparse Tables (tutor10)

This model shows sparse data tables based on expressions. **Problem**
The complete model code in LPL for this model is as follows:

```
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 FROM 'tutor.txt' '%1:Table';
READ '%1' : ROW{i} (i , Ordered , Price);
READ '%2' : ROW{j} (j , Capacity);
READ '%3' : COL{j} j , ROW{i} ( i , COL{j} Hours[i,j] );

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;
WRITE profit, Robots, Steps, Hours;
END
```

LPL Modeling Steps

Sparsity is a tables is a very important concept, it means that only a subset of the Cartesian product of table elements are used . 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

        VARIABLE Robots{i | Ordered >= 5};

means that a variable `Robots` is declared for every element in `i` such that the order is larger then 5. In mathematical notation:

$$x_i, \forall \{i | i \in I = \{1 \dots n\}, O_i \geq 5\}$$

In our case it means that the two variables `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:

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

means that there exists a value within table `Hours` for each combination of (i, j) with $i \in I$ and $j \in J$ such that $i \neq 10$ or $j \neq 4$.

3. The condition `i<>10 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 `~(i=10 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 write the maximizing function then as:

```
MAXIMIZE profit: SUM{i | Price>100} Price * Robots;
```

Question 10 *(Answer to 10)*

1. *How does the model change, if one modify the condition* `Ordered>5` *to* `Ordered>6` *in the variable declaration?*

2. *Change the constraint definition* `Steps{j}:...` *to* `Steps{j|j<=3}....` *Analyse the solution.*

Answer 10 *(Question of 10)*

1. *The variable* `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* `step4` *to* `step8` *are not "tight". This can also be seen by showing the reduced values of the constraints.*

12 Predefined Functions (tutor11)

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

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR11 "Predefined Functions";
  OPTION randSeed:=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} (Price?>100) * Robots;
    (* another formulation is:
        profit: SUM(i) IF(Price<100,100,Price) * Robots; *)
  WRITE 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 `OPTION randSeed:=...`.

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. The `?>` operator returns the larger of the two operands. Hence, `12?>10` in LPL means `12`. The *smaller of*-operator also exists, it is `?<`.

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

```
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.

5. Other functions are available, such as `abs()`, `ceil()`, `sin()`, `cos()`, `log()`, and others.

Question 11 *(Answer to 11)*

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.*

Answer 11 *(Question of 11)*

1. *The assignment is* `Price{i} := RNDN(100,20);` *The function* `RNDN` *returns a normal distributed value.*

2. *The expression is* `SUM{i|Price>120} Price` *. There is another way to express the same:* `SUM{i} IF(Price>120,Price)` *.*

13 Index Operators (tutor12)

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

$$\sum_{i \in \{1...1000\}} x_i$$

which is a shortcut for

$$x_1 + x_2 + x_3 + \ldots + x_{999} + x_{1000}$$

The operator $\sum$ is an *indexed operator* for summation terms over a set. There exist other such operators, for example max (returning the largest element in a list) or argmin (returning the list position of the minimal element). In LPL, they can be used in the same way.

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR12 "Index Operators";
  OPTION randSeed:=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} ALIAS H := TRUNC(RND(0,9));

  VARIABLE
    Robots{i|SUM{j}H>24 AND (FORALL{j}(H<8) OR EXIST{j}(H=0))};
  CONSTRAINT
    Steps{j}: SUM{i} Hours * Robots <= Capacity;
    Order{i}:  Robots  >=  Ordered;

  MAXIMIZE profit: SUM{i} Price * Robots;
  WRITE profit, Robots;
  WRITE{i}: SUM{j}H>24;
  WRITE{i}: FORALL{j}(H<8);
  WRITE{i}: EXIST{j} (H=0);
  WRITE: ATLEAST(5){i,j} (H>7);
END
```

LPL Modeling Steps

In various location of the model we use index operators.

1. The expression `SUM{i} Price*Robots`, for example, means $\sum_i Price_i \cdot Robots_i$.

2. Two other operators are used: `FORALL` and `EXIST`. We now explain the expression containing them.

3. `SUM{j} H`, calculates the number of hours used to produce each Robot i ($\sum_j H_{i,j}$). It returns a value for each i. `SUM{j} H > 24` returns for each i true or false, depending on whether the sum is larger than 24. (This is the case for all $i \neg 10$.)

4. `FORALL{j} (H<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 $i\, in\, \{2, 3, 7, 9, 10\}$.)

5. `EXIST{j} (H=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 $i\, in\, \{2, 3, 4, 6, 7, 10\}$.)

6. Hence, the expression in the variable declaration `Robots{i| ...` defines a subset of `i`, and for each element in that subset a variable instance is created. In our case, the subset is $\{2, 3, 4, 6, 7, 9\}$. Hence, six variable instances `Robots` are created and not 10. It follows that a subsequent expression `SUM{i} Robots[i]` only sums over 6 (not 10) variable instances. The expression is, therefore, equivalent to

   ```
   Robots[2] +Robots[3] +Robots[4] +Robots[6] +Robots[7] +Robots[9]
   ```

Question 12 *(Answer to 12)*

1. *What is returned by* `MAX{i} Price`*?*

2. *What is returned by* `ARGMIN{i} Price`*?*

3. *What means* `MAX{i} MIN{j} H`*?*

4. *What means* `ATLEAST(5){i,j} (H>7)`*?*

Answer 12 *(Question of 12)*

1. 191. *This is the largest value in the price list.*

2. 4. *The smallest price value (105) is at the fourth position within the price list.*

3. *It means "choose the smallest value in each row* `i` *of the matrix* `H{i,j}` *and from the obtained list choose the largest value". It is 3.*

4. *It means that "at least five value in the table* `H` *are greater than 7". This expression returns 1 (one) or 0 (zero) depending on whether it is true or false. In our case, it is true because 7 elements have value 8.* `ATLEAST` *is also an index operator that needs an additional numerical parameter (here 5).*

14 Expression Evaluation ([tutor13](#))

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

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR13 "Expression Evaluation";
  OPTION randSeed:=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} ALIAS H := TRUNC(RND(0,9));

  PARAMETER
      x{i} := MIN{j|H} H;
      y{i} := MAX{j} H;
      MinCapH{i} := MIN{j|H} Capacity/H;
  WRITE '
      Maxi(x)   Mini(y)   Min Capa/Hour
      -------------------------------
  %7s    %3d       %3d         %4d\n'
    : ROW{i}(i, x , y , MinCapH);
END
```

LPL Modeling Steps

This model declares some tables the values in which are randomly generated.

1. `MIN` and `MAX` are two index-operators which returns the minimum or maximum of a list of expressions.

2. The expression `MIN{j|H} H` returns for each `i` the smallest value different from zero in H over `j`. (In this context, `..|H` means `..|H<>0`.) Note that the condition is necessary in the definition of `MinCapH` to avoid a "division-by-zero" error.

3. `ROW` is another index operator in the context of output (using here in a `WRITE`). It says to output the four elements (`i`, `x`, `y MinCapH`) line by line over `i`.

1. *Simplify the expression* `MAX{i} MAX{j} Hours[i,j]`.

2. *Run the model and see the output. Then change* `i:=/1:10/` *to* `i:=/1:100/` *and run the model again. What do you notice.*

1. *The two indexes can be combined into one, One can use the alias* `H`, *and the index-list* `[i,j]` *is not necessary. Hence:* `MAX{i,j} H`.

2. *Well, 100 lines instead of just 10 are written.*

15 Goal Programming (tutor14)

LPL can be used to model soft constraints using goal programming. **Problem**
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR14 "Goal Programming";

  VARIABLE Marie; Jules;
     PTesting; Nprofit; Pprofit; PComp; --slack variables
  PARAMETER dP:=18500  "desired profit";
  CONSTRAINT
    Components: 5*Marie + 5*Jules  <=  350;
    Mounting:   4*Marie + 8*Jules   <=  480;
    Testing:    6*Marie + 2*Jules -PTesting  <=  300;
    Order1:       Marie              >=  20;
    Order2:               Jules    >=  15;
    profit: 300*Marie + 200*Jules -Pprofit+Nprofit = dP;
    --TestBound: PTesting=10;

  MINIMIZE deviation: Nprofit + PTesting + Pprofit;
  --MINIMIZE deviation: Nprofit + 200*PTesting + Pprofit;
  WRITE deviation, Marie, Jules, Nprofit, PTesting;
  WRITE :profit.lhs-Nprofit;
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 about 300.

2. Furthermore, we will be happy with a profit of 'about' 18500.

3. This problem would be infeasible as we know from problem tutor02 (since the maximizing profit 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 to attain our objective.

5. In the profit constraint, we add two slack variables to cover a positive or negative deviation (`Pprofit` and `Nprofit`).

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

7. The deviation of the positive `PTesting` and the negative `Nprofit` 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 profit of 18250.

Question 14 *(Answer to 14)*

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 as to make `TPesting` as small as possible.*

Answer 14 *(Question of 14)*

1. *The profit cannot be augmented over 19500, even if the "desired profit" `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.lpl` 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.*

16 Loop Programming (tutor15)

This model is the answer to the first question in the model `tutor14.lpl`: **Problem** How to implement a sequence of optimisation. We call this also "parameterized optimisation".

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR15 "Loop Programming";

  VARIABLE Marie; Jules;
    PTesting; Nprofit; Pprofit; --slack variables
  PARAMETER dP:=18500  "desired profit";
  CONSTRAINT
    Components: 5*Marie + 5*Jules    <=  350;
    Mounting:   4*Marie + 8*Jules    <=  480;
    Testing:    6*Marie + 2*Jules -PTesting  <=  300;
    Order1:       Marie              >=  20;
    Order2:               Jules     >=  15;
    profit: 300*Marie + 200*Jules -Pprofit+Nprofit = dP;
    --TestBound: PTesting=10;


  SET i :=/1:9/;    --loop set
  FOR{i} DO         --begin loop
    MINIMIZE deviation: Nprofit + PTesting + Pprofit;
    WRITE TO i&'.sns';
    dP := dP+200;
    --dP:= TRUNC(1.1*dP);
  END               --end loop


   --reading all snapshot files automatically
  FOR{i} DO READ FROM+ i&'.sns'; END
  WRITE dP, Marie, Jules TO 'tutor15.nom';
END
```

LPL Modeling Steps

The only modification takes place at the `MINIMIZE` function. It is embedded in a loop.

1. First we add a set `i` the cardinality of this set (here 9) 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, we `MINIMIZE` to deviation in the same way as in the single case.

4. Then we save the complete data store together with the solution in a snapshot file. Snapshot file contain (in a readable form) all data. After each optimisation the data are stored in the files `1.sns`, `2.sns` , ... , `9.sns` using the instruction

   ```
   WRITE TO i&'.sns';
   ```

   The expression `i&'.sns'` defines a string concatenating `i` and `'.sns'`. Since the file names have extension "sns", LPL interprets this as a "write a snapshot".

5. After the `WRITE`-instruction the desired profit `dP` is modified and the loop is repeated.

6. After the first loop, a second loop is executed, which reads all snapshot file at once. The last `WRITE` shows three tables `dP`, `Marie` and `Jules`. The values are acumulated and shown the "progress" of the result in the optimization loop.

Question 15 *(Answer to 15)*

1. *What happens if the capacity of* `Mounting` *is extended? Can we make more profit?*

2. *We would like to rise* `dP` *by 10% within the loop. How can we do this?*

Answer 15 *(Question of 15)*

1. *Nothing happens, the profit is the same. This is because* `Mounting` *is never the limiting constraint.*

2. *Replace the instruction* `dP:=dP+200;` *with* `dP:= TRUNC(1.1*dP);` *and run again.*

17   Logical Constraints (tutor16)

We include some logical constraints.   **Problem**

The complete model code in LPL for this model is as follows:

```
MODEL TUTOR16 "Logical Constraints";

  SET  i := / 1:10 /;
  INTEGER VARIABLE  Robots{i} [0,500];
  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 ];
  CONSTRAINT
    Components: SUM{i} HC[i]*Robots[i] <=  3500;
    Mounting:   SUM{i} HM[i]*Robots[i] <=  4800;
    Testing:    SUM{i} HT[i]*Robots[i] <=  3000;

    Log1:     Robots[2]>=150 -> Robots[6]>=30;
    Log2:     ATLEAST(5){i} (Robots[i]>=20);
    --Log3:    Robots[1]>=10 and Robots[3]>=15;
    --Log4:    NOR{i} (Robots>100);
  MAXIMIZE profit: SUM{i} Price[i]*Robots[i];
  WRITE profit, Robots, HC, HM, HT;
END
```

LPL Modeling Steps

The two additional logical constraints are `Log1` and `Log2`. Removing both constraints and solving the model produces the following solution: $Robots_2 = 300, Robots_9 = 480$ the profit is $profit = 252000$. Let's call this the *basis solution*.

1. Add the `Log1` constraint now. It says that if `Robots[2]` is equal or larger than 150 (this is the case in the basis solution) then `Robots[6]` must be larger or equal to 30 (this is not the case in the basis solution). Solving the problem with the additional constraint `Log1` says that the

profit drops to 244600 and `Robots[6]` goes up to 30 as required by the constraint.

2. The constraint `Log2` imposes that at least 5 different types of Robots must be produced with at least the quantity of 20 pieces. Solving the problem imposes now that 5 types of Robots are produced as can be verified.

Question 16 *(Answer to 16)*

1. *Add the constraint* `Log3`. *How does the solution change.*

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

Answer 16 *(Question of 16)*

1. `Robots[1]` *and* `Robots[3]` *enter the solution as required. However* `Robots[2]` *drops to 149. Therefore,* `Log1` *has no effect anymore and* `Robots[6]` *now is zero.* `Log3` *is not – strictly speaking – a logical constraint, it just impose two lower bounds. On the other side, this means that a list of constraints can be expressed by a single constraint linking them together with* `AND`.

2. `CONTRAINT Log4{i}: Robots<=100;` *another way to formulate it 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`. `NAND` *means "not all". By the way, LPL translates the "logical" constraint automatically into ordinary variable bounds (see EQU-file).*

18 Calling Submodels (tutor17)

This model shows how a larger model can be broken down into logical units. **Problem**
A model can be subdivided hierarchically into submodels. They can be
executed separately by calling them. In our case, we use submodels to define
various data sets for the same main model.
The complete model code in LPL for this model is as follows:

```
MODEL TUTOR17 "Calling Submodels";
  ---- main LP model
  SET   i; j;
  PARAMETER Ordered{i}; Price{i}; Capacity{j}; Hours{i,j};
  VARIABLE  Robots{i};
  CONSTRAINT
    Steps{j}: SUM{i} Hours*Robots <= Capacity;
    Order{i}: Robots >= Ordered;
    profit FREEZE: SUM{i} Price*Robots;


  ---- execution block
  PARAMETER x:=0;
  WRITE '---run myData1\n';
  myData1;          -- call submodel
  WHILE x<4 DO      -- loop (4 times)
    MAXIMIZE obj: profit;
    WRITE 'profit: %8d\n': profit;
    lowerPrice;     -- call submodel
    x:=x+1;
  END

  WRITE '---run myData2\n';
  EMPTY TUTOR17;    -- clear the data store
  IF profit<160000 THEN
    myData1;
    MAXIMIZE obj1: profit;
    WRITE '(myData1) profit: %8d\n': profit;
  ELSE
    myData2;
    MAXIMIZE obj2: profit;
    WRITE '(myData2) profit: %8d\n': profit;
  END;
```

```
WRITE '---run myData3\n';
EMPTY TUTOR17;
myData3;
MAXIMIZE obj3: profit;
WRITE '(myData3) profit: %8d\n': profit;


WRITE '---run DBdata\n';
EMPTY TUTOR17;
DBdata;
MAXIMIZE obj4: profit;
WRITE '(DBdata) profit: %8d\n': profit;




------ submodel declarations ------
MODEL myData1 "First Data Set";
  READ FROM 'tutor.txt' '%1:Table';
  READ '%1': ROW{i} (i , Ordered , Price);
  READ '%2': ROW{j} (j , Capacity);
  READ '%3': COL{j} j, ROW{i} (i,COL{j} Hours);
END

MODEL myData2 "Second Data Set";
  READ FROM 'tutor1.txt' '%1:Table';
  READ '%1': ROW{i} (i , Ordered , Price);
  READ '%2': ROW{j} (j , Capacity);
  READ '%3': COL{j} j , ROW{i} (i,COL{j} Hours);
END

MODEL myData3 "Randomly Generated Data Set";
  OPTION randomSeed := 1;
  i         :=  /1:100/;        -- 100 products
  Ordered{i} :=  RND(0,10);
  Price{i}   :=  RND(100,300);
  j         :=  /1:20/;         -- 20 processes
  Capacity{j}:= RND(5000,20000);
  Hours{i,j} := IF(RND(0,1)<=0.3,RND(1,15));
END

MODEL DBdata "Read Data from a DB";
  STRING PARAMETER DB:='tutor18a.mdb';
  READ{i} FROM 'DB,Robot':
    i='robot', Ordered='ordered', Price='price';
```

```
   READ{j} FROM 'DB,Process':
     j='processes' , Capacity='capacity';
   READ{i,j} FROM 'DB,Hours':
      i='rID', j='pID', Hours='hours';
  END

  MODEL lowerPrice "a lower price scenario";
    Price{i} := 0.9*Price;
    --Capacity{j} := 1.1*Capacity;  --2nd question
  END
 END
```

LPL Modeling Steps

There is a main model is just an LP model without a objective function and
without data. It consists of the first 8 code lines. Then follows an execution
block. Finally, several submodels are declared. Note that the submodels
are declared **after** they are used (called) in the execution block of the main
model. Normally, an identifier must be declared *before* its use. But submodels
can also be declared after their use (call). This allows the modeller to write
the code in a clean top-down fashion, by refining the "calls" *afterwards*.

1. The execution block first declare a parameter x which is used as loop
   parameter. Then a string is written to the output.

2. The next instruction myData1; calls the submodel *myData* which is
   executed at this point. It assigns the data to the model by reading
   them from a file tutor.txt.

3. Next the WHILE-loop is entered and the main model is solved. (Note
   that because of the FREEZE keyword profit is not considered as a
   constraint. It is activated as expression of the objective function.)

4. Next the profit is written to the output and the price vector is lowered
   by 10% by calling the submodel lowerPrice. x is advance and the loop
   is repeated (4 times).

5. Next we clear all data of the model by the EMPTY instruction.

6. Then profit<160000 is checked. This is not the case, so the ELSE-part
   is executed.

7. The `ELSE`-part calls the model `MyData2`, that is, it reads the data from another file (`tutor1.txt`).

8. Next the `profit` is maximized by solving the model and the result is written to the NOM-file.

9. After that the data store of LPL is cleared again another data set is read by executing `myData3` and the model is solved again.

10. This is repeated again, this time by reading the data from a database (running `DBdata`).

Question 17 *(Answer to 17)*

1. *Place an `END` just before the submodel declarations. What happens when the code is executed?*

2. *In the first loop the price are lowered by 10% and consequently the profit falls by 10%. How could this be compensated?*

Answer 17 *(Question of 17)*

1. *An "Unresolved forward reference" error occurs. The submodels called in the main model are not declared. (remove the added `END`).*

2. *By expanding the capacity by 10%. Add the instruction*

```
Capacity{j} := 1.1*Capacity;
```

in the model `lowerPrice` and run the model. The profit is stable and keeps its value around 230000.

19   Link to Database (tutor18)

This models shows how the data can be read from and write to a database.   **Problem**
The complete model code in LPL for this model is as follows:

```
MODEL Tutor18 "Link to Database";

  PARAMETER para := 5;
  STRING DB := 'tutor18a.mdb'  "My ACCESS database";

  SET i; j; h1{i,j}; ii{i};
  PARAMETER a{i}; c{j}; h{h1};
  STRING PARAMETER b{i};
    p{j}; nII{ii};

  READ{i} FROM 'DB,SELECT * FROM Robot WHERE ID<=:para' :
    i='ID', b='Robot', a='Price';
  READ{j} FROM 'DB,Process' :
    j='ID', p='Processes', c='capacity';
  READ{i,j} FROM 'DB,Hours' :
    i = 'rID', j='pID',
    h1= ('rID','pID'), h='Hours';
  READ{i} FROM 'DB,Rob' :
    i= 'ID', ii='ID' , nII='Robot';

  WRITE i,a,b , j,c,p , h1,h , ii,nII;
  WRITE{j} TO- 'DB,NEW': 'ID'=j, 'aa'=p, 'num'=c, 'xx'=j;

END
```

LPL Modeling Steps

Data can be read from and written to databases using `READ` and `WRITE` instructions.

1. The `FROM` string contains a database name or a `STRING PARAMETER`
   name (here `DB`) to identify the database, followed by a table name,
   a query name or a parameterized SQL `SELECT` statement. Database
   name and table name must be separated by a comma.

2. The parameter `para` in the SQL statement must begin with a colon (:). LPL replaces it with the content of the parameter (here `:para` is replaced by `5`). Hence, the query in the first READ statement will be:

```
SELECT * FROM Robot WHERE ID<=5
```

3. The expression following `:` contains a list of mapping: `i='ID'`, for example, means that the field `ID` in the database table must be read and assigned to the set `i` in LPL.

4. The `WRITE` contains a `TO` attribute. It also contains the database name and a table name. (Here the database name is `DB` and the table name is `NEW`.)

5. The minus after `TO` means that the content of the table must be cleared before the first record is written.

One can also easily create a database using `lplw.exe`. This can be done as follows:

Comments

1. Load the model `tutor18.lpl` into `lplw.exe`. Run the model.

2. Then click the menu "Tools/Create SQL-files". This creates two files: (1) `tutor18.sql`: This is a SQL-script that can be executed by a database server. (2) `tutor18.sq2`: This file is a LPL DATA model which contains all `READ/WRITE` instructions to read/write from the database.

3. The click the menu "Tools/Create ACCESS DB". A database file `tutor18.mdb` will be created. A message box indicates the creation.

4. Open then database `tutor18.mdb` using MS Access. It has created the tables according to the SQL script generated in the previous step.

Question 18 *(Answer to 18)*

1. Change the string `'tutor18a.mdb'` *to* `'tutor.mdb'` *and run again. What happens?*

Answer 18 *(Question of 18)*

1. An error is produced: "434 Database file not found" and the run is aborted.

20   Create a Database (tutor19)

This models shows how an LPL code can create a complete database.   **Problem**
   ⟦To understand this example you need to know database technology.⟧
The complete model code in LPL for this model is as follows:

```
MODEL Tutor19 "Create a Database";

  STRING PARAMETER DB:='tutor19a.mdb' "database name";

  SET i := /1:9/;   j := /A B C D E/;
    ij{i,j} := /1 B, 2 C, 4 E, 5 A, 6 D, 3 B, 7 D/;
  PARAMETER a{j} := j*j;  d{j} := [3 2 4 5 1];
    b{i,j} := i*j;
  STRING c{j} := ['AA', 'BB', 'CC', 'DD', 'EE'];

  WRITE TO** 'DB,table1' : 'fieldA' = 'line1' , 'f2' = 234.78;
  WRITE TO+  'DB,table1' : 'fieldA' = 'line2' , 'f2' = 23.709;

  WRITE{j} TO* 'DB,table2' :  'j' = j , 'a' = a , 'c' = c;
  WRITE{i} TO* 'DB,table3' : 'ID' = i , 'j'=COL{j} b;
  WRITE{j} TO* 'DB,table4' : 'ID' = j , 'd'=d;

  WRITE{j} TO* 'DB,table5' : '_d'=d , 'j' = j , 'a'=a;

  WRITE{ij[i,j]} TO* 'DB,table6' :
    'ID_table3_ID'=i, 'ID_table4_ID'=j;
  END
```

LPL Modeling Steps

This LPL code creates a complete database. The database name is defined
in parameter DB ('tutor19a.mdb').

1. The WRITE TO** create the database, if it exists already it will be
   deleted. At the same time a table table1 within DB is created.

2. The fields in table1 are named fieldA and f2. One record is added
   to the table.

3. The `TO+` will add records to an already created table (`table1`). The fields must correspond. Hence, the second `WRITE` add another record to tha table.

4. The third `WRITE` creates a second table (`table2`) within the database. This is forced by the `TO*` (one star after the keyword `TO`). In this way we can add any table to an existing database, provided that the table does not exist already.

5. The fields are `j`, `a`, and `c`. Since the statement is indexed over `j`, five records will be added to the table.

6. The fourth `WRITE` instruction creates a third table: `table3`, with the field names `ID` and `j1` to `j5`. The `COL{j}` keyword has a special meaning in this context: create as many fields as `j` has elements (5), and name the fileds by concatenating 'j' with the numbers '1' to '5'. Since the statement is indexed, 9 records are added and the matrix `b{i,j}` is stored in table `table3`.

7. There is a particularity about the field `ID`. The translator interprets a fieldname beginning with a capital **I** (which does not contain a `_` character, see below) as a primary key field. Hence a primary key on field the `ID` is created in addition. The same is valid for the creation of table `table4`. Again a primary key is generated on field `ID`.

8. The sixth `WRITE` statement adds the table `table5`. It is similar to table `table2`. However, it contains a field `_d`. A fieldname beginning with the character `_` instructs the translator to generate also an INDEX on this field. Indeed, opening the table `table4` with the database server reveals that the records are sorted according to `d{i}`.

9. The last `WRITE`-statement creates the table `table6` with two fields named `ID_table3_ID` and `ID_table4_ID`. A field beginning with a character **I** and having two `_` character – hence having the syntax `Ix_y_z` , where `x`, `y`, and `z` are arbitary strings without an underscore char – are interpreted as foreign keys fields. In this case `y` is interpreted as the foreign table and `z` as the foreign primary key field. In SQL syntax, the translator will execute the following statement while creating the table `table6`:

```
    CREATE TABLE table6(
      ID_table3_ID VARCHAR(150),
        FOREIGN KEY (ID_table3_ID) REFERENCES table3(ID),
      ID_table4_ID VARCHAR(150),
        FOREIGN KEY (ID_table4_ID) REFERENCES table4(ID));
```

There is another simplified way to create a database from an LPL model. **Comments**
(Only possible with LPL's Enterprise Package.) As an exercise – if you have
LPL Entreprise Package:

1. Open `lplw.exe` with the model `tutor08.lpl`. Then run the model
   (Menu: 'Run/Run Model').

2. Now choose menu: 'Tools/Create SQL files'. This menu point will
   generate two files: `tutor08.sql` and `tutor08.sq2`. The firts is an
   SQL-script that can be executed by most database server to create a
   complete database. The second file is a list of LPL instructions that
   can be included into the LPL-model code to read the data from this
   database.

3. Then chosse menu: 'Tools/Create ACCESS DB'. This will automati-
   cally run the script generated before and create a complete Microsoft
   ACCESS database.

21 Write-Format Examples (tutor20)

This model shows all output formatting with the WRITE statement as ex- **Problem**
plained in the reference manual in Chap. 9.2.2.
The complete model code in LPL for this model is as follows:

```
MODEL tutor19 "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 (hexadeci)     : %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 'n (1 decimal)    : %7.1n\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';

  -- All date/time format specifiers

  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 (month) : %tA\n' : d;
    write 'ta (month) : %ta\n' : d;
    write 'tY (month) : %tY\n' : d;
    write 'ty (month) : %ty\n' : d;
    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;
END
```

22 Drawing Library I (tutor21)

This model shows how to use the graphic library of LPL. The complete model code in LPL for this model is as follows:

```
model draw "Drawing Library I";
  Draw.Ratio(3,3,480);
  Draw.Rect(0,0,150,150,2);
  Draw.Line(5,5,5,150,0);
  Draw.Line(5,5,150,5,0);
  Draw.Line(5,75,75,5,0);
  Draw.Line(5,150,50,5,0);
  Draw.Text('A',2,2,0);
  Draw.Text('B',2,75,0);
  Draw.Text('C',40,42,0);
  Draw.Text('D',51,6,0);
  Draw.Text('y',2,145,0);
  Draw.Text('x',145,0.9,0);
  Draw.Ellipse(37.5,39.5,40.5,42.5,0);
  Draw.Text('6x+2y = 300',27,100,-75,16,0);
  Draw.Text('5x+5y = 350',50,35,-45,16,0);
end
```

1. All instructions to draw a picture begin with 'Draw.'. They draw sequentially into the same picture, which then is stored automatically to the file abc.jpg at the end of a model run.

2. Draw.Ratio stretches all x- and y-coordinates by a multiplier. For example, Draw.Ratio(3,3); instructs the drawing library to interpret each subsequent (x, y)-coordinate as $(3 \cdot x, 3 \cdot y)$.

   Normally, the coordinate point $(0, 0)$ is the left-top of the picture. A third parameter in the Ratio function, sets the $(0, 0)$ point at the left-bottom. Hence Draw.Ratio(3,3,160); put the point $(0, 0)$ at $(0, 480)$.

3. Draw.Rect(0,0,150,150,2); draws a rectangle from $(0, 0)$ (left-bottom) to $(300, 300)$ (right-top) with color code 2 (light blue).

4. Draw.Line(5,5,5,150,0); draws a line from $(15, 15)$ to $(15, 450)$ with color code 0 (black).

5. Draw.Text('A',2,2,0); draws the text 'A' at point $(6, 6)$.

6. `Draw.Ellipse(37.5,39.5,40.5,42.5,0);` draws an ellipse in the rectangle from $(112.5, 118.5)$ to $(121.5, 127.5)$ with color-fill code 0 (black).

7. `Draw.Text('6x+2y = 300',27,100,-75,16,0);` draws the text '6x+2y = 300' at point $(81, 300))$ at an angle of -75^0 with height 16 and color code 0.

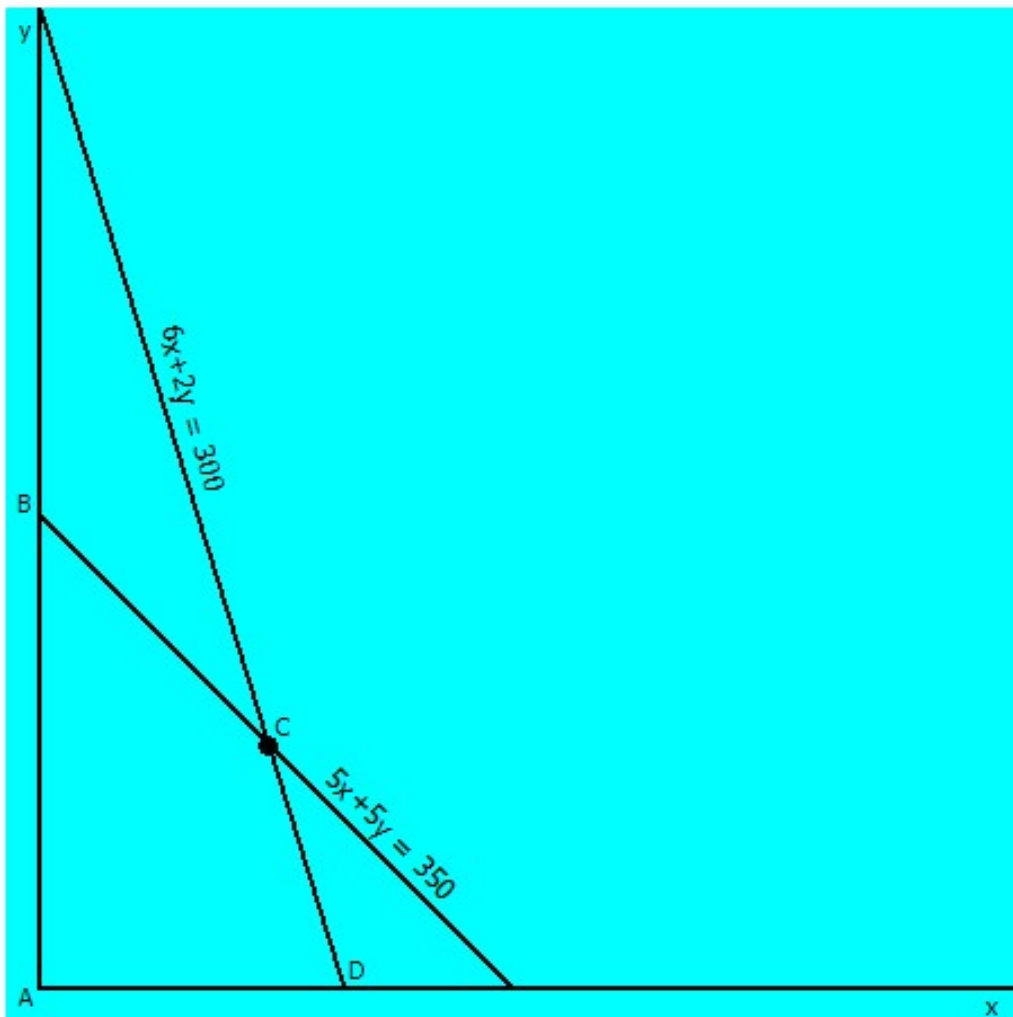The result is stored in file `abc.jpg`. The picture format is jpeg. It is displayed in Figure 2.

Figure 2: The Picture

23 Drawing Library II (tutor22)

Like tutor21.lpl, this model shows how to use the graphic library of LPL. The complete model code in LPL for this model is as follows:

```
model colors "Drawing Library II";
  if 1 then FirstExample; else SecondExample; end

  model FirstExample;
    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.Ratio(20,20);
    for{y,x} do Draw.Rect(x,y,x+1,y+1,a); end
    --- using RGB function for collos.
    for{i={1:256}} do
      Draw.Rect((i-1)/8,10,i/8,12,RGB(0,0,i-1)); end
    for{i={1:256}} do
      Draw.Rect((i-1)/8,13,i/8,15,RGB(i-1,i-1,255)); end
  end

  model SecondExample;
    Draw.Ratio(2,2);
    set i alias j := /1:100/;
    --integer parameter c{i,j} := (i^2+2*j)%32+64;
    --integer parameter c{i,j} := ((i/10)^2+2*j)%32+64;
    integer parameter c{i,j} := (sqrt(10*i*j))%32+128;
    for{i,j} do Draw.Rect(i,j,i+1,j+1,c); end
  end
end
```

1. The model executes the submodel `FirstExample` or `SecondExample` depending on whether the `if` is true or false (here `FirstExample` is executed only by default.

2. `for{y,x}` means to excecute `Draw.Rect(x,y,x+1,y+1,a);` $8 \cdot 32$ times with the corresponding parameters.

3. Using the `RGB(r,g,b)` function, one can generate a RGB color with the red (r), green (g), and blue (b) part.
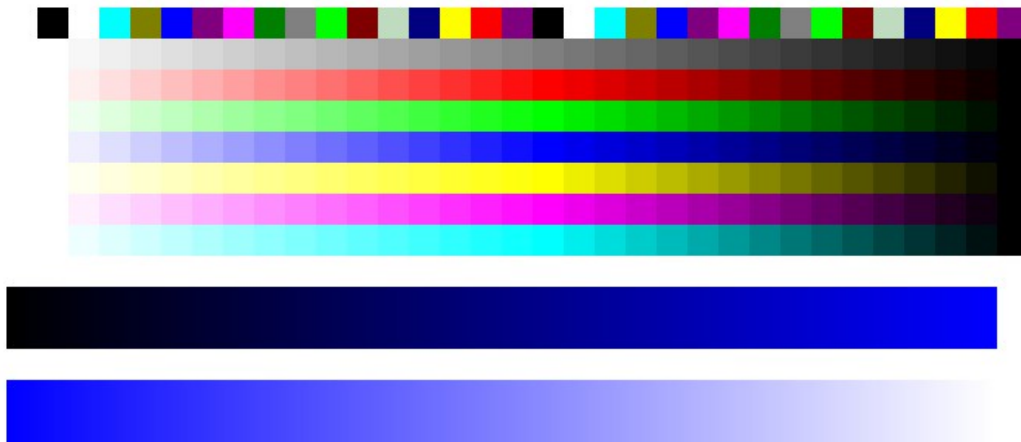
Figure 3: The Picture drawn by FirstExample
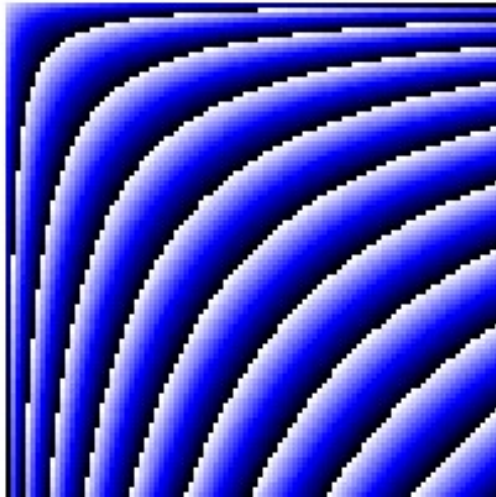
The pictures of the two models are shown in Figure 3 and 4

Figure 4: The Picture drawn by SecondExample

24   Conclusion

This ends the tutorial of LPL. Other introductional papers as well as the reference manual can be found at LPL Documentation.

References

[1] HÜRLIMANN T., Reference Manual for the LPL Modelling Language, (see www-virtual-optima.com).