The Usage of Sets

Usually, with a small amount of decision variables and limited constraints, we can use the traditional way and enter the objective function and the constraints one by one. However, if we meet the problem with a series of similar constraints and express long, complex formulas, the traditional way would be troublesome, and the usage of sets would be a great choice. Sets allow to express the large models very quickly and easily. Usually, in large models, you'll encounter the need to express a group of several very similar calculations or constraints.

LINGO recognizes two kinds of sets: primitive and derived. A primitive set is a set composed only of objects that can't be further reduced. A derived set is defined using one or more other sets. In other words, a derived set derives its members from other preexisting sets.

Now, let me introduce the primitive sets, which will also be used in our problem. A primitive set definition has the following syntax:

setname [/ member_list /] [: attribute_list];

The three words are corresponding to the name of the set, its members (objects contained in the set), and any attributes the members of the set may have. The member list and attribute list are optional. The set section begins with the keyword **SETS:** (including the colon) and ends with the keyword **ENDSETS**. In our case, we have the same type the members: each of the member has 12 values corresponding to 12 months. There's my code:

```
sets:
Month: rlabor, rtons, olabor, otons,limittons, soldtons, demandtons,
pricepertons, storagetons, storageprice;
endsets
```

The rtons, otons, soldtons and storagetons are the decision variables and all other variables are coefficients, which will be set in the following section.

Since the derived sets would not be used in our problem, if you would like to learn more about that, please refer the official Lingo Manual.

Usage of Data

To initialize the members of certain sets and assign values to certain set attributes, LINGO uses a second optional section called the **data** section. The data section allows you to isolate data from the equations of your model. This is a useful practice in that it leads to easier model maintenance and facilitates scaling up a model to larger dimensions.

Similar to the sets section, the data section begins with the keyword **DATA:** (including the colon) and ends with the keyword **ENDDATA**. In the data section, There's the syntax:

object_list = value_list;

The **value_list** contains the values to assign to the objects in the object list, optionally separated by commas. In our problem, from the table, we know that the cost of labor in regular time from month 1 to month 3 is $4, from month 4 to month 9 is $6 and from month 10 to month 12 is $4. So I get the following code:

```
data:
rlabor=4,4,4,6,6,6,6,6,6,4,4,4;
enddata;
```

Similarly, I can assign the initial value for olabor(labor cost in overtime), limittons(limit on raw material availability), demandtons(demand), pricepertons(selling price) and storageprice(storage price). All of them would be assigned with 12 values, which corresponding to the value from month 1 to month 12.

Set Looping Function

Set looping functions allow you to iterate through all the members of a set to perform some operation. There are currently four set looping functions in LINGO. The names of the functions and their uses are following:

Function	Use

@FOR	The most powerful of the set looping functions, *@FOR* is primarily used to generate constraints over members of a set. *@FOR* may also be used in calc sections to assign values to attributes across the members of a set.
@SUM	Probably the most frequently used set looping function, *@SUM* computes the sum of an expression over all members of a set.
@MIN	Computes the minimum of an expression over all members of a set.
@MAX	Computes the maximum of an expression over all members of a set.
@PROD	Computes the product of an expression over all members of a set.

The syntax for a set looping function is:

@function(setname: expression_list);

where @function corresponds to one of the four set looping functions listed in the table above. **setname** is the name of the set you want to loop over.

In our problem, the objective function is the sum of each month's revenue minus the sum of regular time labor cost, minus the sum of each month's overtime labor cost and the sum of each month's storage cost. The revenue in month j is the selling price times the sold tons. Since I've assigned the values to selling price, the sum of each month's revenue is:

```
@sum(month:pricepertons*soldtons)
```

Similarly, we can get the sum of labor cost and storage cost.

Also, to model the constraints, we need to use **@for** function. For the constraint **(1.6)**, which is the sold tons in month j should be less than or equal to the demand in month j. For example, the sold tons in month 1 should be less than or equal to 400. Since I've assigned the value to the demands in the **DATA** section, the constraint is:

```
@for(month(I):
soldtons(I) <=demandtons(I)
);
```

The index **I** is from 1 to 12 since we've assigned 12 values to **demandtons(I)**, which should be the same index. Soldtons(I) means the selling tons in month I. For example, Soldtons(12) means the selling tons in month 12.

Although there're only two loop functions used in our problem, the usage of other two loop function is similar. If you would like to find more example, please refer the official Lingo Manual.

The usage of filter

Sometimes, I only need to loop some parts of the set, not the whole part. For example, for the constraint (1.8), which is that, during the months 4,5,6,7,8,9, the company can hire enough labor to produce 800 tons of products during the regular time.

	R_j<=800, for j=4,5,6,7,8,9	**(1.8)**

In this case, we can use a conditional qualifier (filter) on the set index to accomplish this as follows:

```
@for(month(I)|I #GE# 4 #and# I #LE# 9:
Rtons(I)<=800
);
```

The #LE# or #GE# symbol is called a logical operator. This operator returns true if I is greater than or equal to 4 and less than or equal to 9. Otherwise, it returns false. Therefore, when LINGO loop the function, it plugs the set index variable, I, into the conditional qualifier I #GE# 4 and #LE# 9. If the conditional qualifier evaluates to true, Rtons(I) will be added to the constraints. The logical operators recognized by LINGO are:

#EQ#	equal
#NE#	not equal
#GE#	greater-than-or-equal-to
#GT#	greater than
#LT#	less than
#LE#	less-than-or-equal-to