- Rather than defining the parameter `n` to represent the *number of paint colors*, we have defined the **set** `P` which actually contains the paint colors. This will become clearer when we define the data below.
- Rather than indexing parameters over the indices `1..n`, we have indexed them over the members of the set `P`.

Now, the data can be defined as follows:

```
## Example Two - Data with sets

set P:= blue gold;

param t:= 40;

param p:= blue  10
          gold  15;
param r:= blue  40
          gold  30;
param m:= blue  1000
          gold  860;
```

Note that after the set was defined, each parameter element was stated using the set element name, rather than the index number. As you can see, this is much easier to read and follow. In fact, we can combine the set notation with the more compact data notation introduced in the previous section, and we get:

```
## Example Two - Another way to write the data with sets

set P:= blue gold;
param t:= 40;

param:  p r m:=
blue 10 40 1000
gold 15 30 860;
```

5 Variables and Parameters With Two Dimensions

Often, a mathematical programming model has variables and parameters with two dimensions. We can write the parameter data in table form, and AMPL allows us to enter this information in table form in the data file.

Consider the following example. Berkeley Paint Company has expanded, and now has three warehouses, all filled with blue paint. In a particular week, paint has to be shipped to four customers. For each warehouse and customer, *per gallon* shipping costs are different. Shipping costs are displayed in the following table. The warehouses are in the first column, the customers are in the first row, and the warehouse-customer shipping cost can be read from the table.

	Cust 1	Cust. 2	Cust. 3	Cust. 4
Warehouse 1	1	2	1	3
Warehouse 2	3	5	1	4
Warehouse 3	2	2	2	2

In addition, the supply of blue paint at each of the warehouses is:

Warehouse 1	250
Warehouse 2	800
Warehouse 3	760

The demand at each customer is:

Customer 1:	300
Customer 2:	320
Customer 3:	800
Customer 4:	390

Note that total supply equals total demand. The following AMPL model is set up to minimize total cost subject to meeting demand. We save the file on the `z:` drive as `z:\myFiles\examp_3.mod`.

```
## Example Three - Model

param warehouse;   # number of warehouses
param customer;   # number of customers
 #transportation cost from warehouse i
 #to customer j
param cost{i in 1..warehouse, j in 1..customer};
param supply{i in 1..warehouse};    #supply at warehouse i
param demand{i in 1..customer};    #demand at customer j

var amount{i in 1..warehouse, j in 1..customer};

minimize Cost:
    sum{i in 1..warehouse, j in 1..customer} cost[i,j]*amount[i,j];
subject to Supply {i in 1..warehouse}:
    sum{j in 1..customer} amount[i,j] = supply[i];
subject to Demand {j in 1..customer}:
    sum{i in 1..warehouse} amount[i,j] = demand[j];
subject to positive{i in 1..warehouse, j in 1..customer}:
    amount[i,j]>=0;
```

Note the following things about this model:

- Each parameter and variable has a longer and more descriptive name.
- Parameters and variables with double indexes (cost and amount) are indexed exactly as before, except that two indexes are given, separated by a comma.
- The objective `Cost` and the constraints `Supply` and `Demand` are capitalized so that AMPL doesn't confuse them with the parameters `supply`, `demand`, and `cost`.

To complete the model, we need to specify the data. The data file is saved on the `z:` drive as `z:\myfiles\examp_3.dat`.

```
## Example Three - Data

param warehouse:= 3;
param customer:= 4;
param cost:     1     2     3     4 :=
       1        1     2     1     3
       2        3     5     1     4
       3        2     2     2     2;

param supply:= 1 250   2 800   3 760;
param demand:= 1 300   2 320   3 800   4 390;
```

Pay particular attention to the way the `cost` parameter data is entered. Note the colon after the the parameter name, and the colon and equal sign after the final warehouse index. Also, note that the the first index (warehouses) is in the first column, while the second (customers) is in the first row, exactly like the table in which we first specified the data.

These model and data files are loaded and executed exactly as before. Note that if you enter `display amount;` after the model has run, AMPL displays all of the values of the **amount** variable. If the program settings are correct, **amount** will be displayed in tabular form.

Also, we could use the set notation introduced in the previous section to clarify the model and data. This is especially helpful if the customers and warehouses have names or location names. For example, a possible model and data follow:

```
## Example Three - Model using sets

set Warehouses;
set Customers;
 #transportation cost from warehouse i
 #to customer j
param cost{i in Warehouses, j in Customers};
param supply{i in Warehouses};    #supply at warehouse i
param demand{j in Customers};    #demand at customer j

var amount{i in Warehouses, j in Customers};

minimize Cost:
    sum{i in Warehouses, j in Customers} cost[i,j]*amount[i,j];
subject to Supply {i in Warehouses}:
    sum{j in Customers} amount[i,j] = supply[i];
subject to Demand {j in Customers}:
    sum{i in Warehouses} amount[i,j] = demand[j];
subject to positive{i in Warehouses, j in Customers}:
    amount[i,j]>=0;
```

The data file might now look like:

```
## Example Three - Data with sets

set Warehouses:= Oakland San_Jose Albany;
set Customers:= Home_Depot K_mart Wal_mart Ace;

param cost:     Home_Depot K_mart Wal_mart Ace:=
       Oakland        1      2       1       3
       San_Jose       3      5       1       4
       Albany         2      2       2       2;

param supply:= Oakland     250
               San_Jose    800
               Albany      760;
param demand:= Home_Depot  300
               K_mart      320
               Wal_mart    800
               Ace         390;
```

6 Integer Programming

Often, a mathematical programming model requires that some (or all) variables take on only integral values. Fortunately, AMPL allows us to incorporate this with only a small change in the model. By adding the keyword **integer** to the **var** declaration, we can restrict the declared variable to integral values. Furthermore, by adding the keyword **binary** to the **var** declaration, we can restrict the declared variable to the values **0** and **1**.

To illustrate, consider the following example. Berkeley Paint Company has now expanded the supply capacity at each of its warehouses. However, this expansion has come at a price. In addition to the shipping costs, now the company has to pay a fixed cost for opening a warehouse.

The supply capacity of paint at each of the warehouses is:

Warehouse 1	550
Warehouse 2	1100
Warehouse 3	1060

The cost of opening each of the warehouses is:

Warehouse 1	500
Warehouse 2	500
Warehouse 3	500

Note that available supply now exceeds total demand. The following AMPL model is set up to minimize total cost subject to meeting demand. In fact, this problem is called the Warehouse Location Problem. We save the file on the `z:` drive as `z:\myFiles\examp_4.mod`.

We use the set notation introduced in Section 4 to clarify the model and data. This is especially helpful if the customers and warehouses have names or location names.

```
## Example Four - Mixed-IP model file for the warehouse location problem

set Warehouses;
set Customers;
 #transportation cost from warehouse i
 #to customer j
param cost{i in Warehouses, j in Customers};
param supply{i in Warehouses};    #supply capacity at warehouse i
param demand{j in Customers};    #demand at customer j
param fixed_charge{i in Warehouses};    #cost of opening warehouse j

var amount{i in Warehouses, j in Customers};
var open{i in Warehouses} binary;   # = 1 if warehouse i is opened, 0 otherwise

minimize Cost:
    sum{i in Warehouses, j in Customers} cost[i,j]*amount[i,j]
         + sum{i in Warehouses} fixed_charge[i]*open[i];
subject to Supply {i in Warehouses}:
    sum{j in Customers} amount[i,j] <= supply[i]*open[i];
subject to Demand {j in Customers}:
    sum{i in Warehouses} amount[i,j] = demand[j];
subject to positive{i in Warehouses, j in Customers}:
    amount[i,j]>=0;
```

Note the following changes in this model:

- We introduced the variable **open** to indicate whether the warehouse is open, and defined it to be **0-1**.
- The parameter **fixed_charge** stores the cost for opening the warehouse.
- The total cost now includes both the shipping costs and the fixed charge for opening a warehouse.
- The constraint **Supply** is changed to ensure that a warehouse supplies paint only if it is open.

To complete the model, we need to specify the corresponding data. The data file is saved on the `z:` drive as `z:\myfiles\examp_4.dat`.

```
## Example Four - Data file for the warehouse location problem

set Warehouses:= Oakland San_Jose Albany;
set Customers:= Home_Depot K_mart Wal_mart Ace;

param cost:     Home_Depot K_mart Wal_mart Ace:=
       Oakland        1       2       1       3
       San_Jose       3       5       1       4
       Albany         2       2       2       2;

param supply:=        Oakland     550
                      San_Jose    1100
                      Albany      1060;
param demand:=        Home_Depot  300
                      K_mart      320
                      Wal_mart    800
                      Ace         390;
param fixed_charge:= Oakland     500
                      San_Jose    500
                      Albany      500;
```

7 Nonlinear programming

Many mathematical programs include nonlinear functions in the constraints and for the objective. Fortunately, AMPL allows us to model and solve nonlinear programs, as well.

First, you have to change the solver, since **cplex** does not solve nonlinear programming problems. For nonlinear problems, we shall use the solver **minos**. So type:

```
option solver minos;
```

To illustrate, let us consider the example of **Portfolio Optimization**. Suppose that we have a set of alternate investments **A**, and that we know the expected rate of return **R** for each of these investments for a set of years **T**. From this data, we can calculate the covariance matrix for the investments.

We would like to