Arrays, Reading to Input Failure, Decisions, Functions

This programming assignment uses many of the ideas presented in sections 3 through 8 of the course notes, so you are advised to read those notes carefully as well as the following program specification.

The Program Specification: Parcel Post

The shipping company Delivering On-time With No Surprises (DOWNS) will ship packages subject to the following constraints:

- The weight of a package must not exceed a specified maximum weight.
- The length of the package must not exceed a specified maximum length. The length of a package is defined to be the length of its longest dimension.
- The size of the package must not exceed a specified maximum size. The size of a package is defined to be the length of the package plus the girth of the package. The girth of a package is the circumference of the package around its two smallest sides and can be calculated using the formula:

```
Girth = 2 * (Dimension1 + Dimension2 + Dimension3 – Length)
```

You need to determine whether certain packages meet these criteria. For example, suppose the maximum weight allowed is 40, the maximum length is 36, and the maximum size is 72. Consider a package with weight 35, and dimensions 12, 16 and 35. The length of the package is 35. The girth of the package is 2*(12 + 16 + 35 - 35) = 2*28 = 56. So the size of the package is 56 + 35 = 91, which is too large.

Input file descriptions and samples:

This program requires the use of two input files. One contains the postal rates and the other contains the transactions. Your program **must** read the postal rates from a file named `parcel.data` and the transactions from a file named `transact.data` — use of other input file names will generally result in a score of zero.

In both input files, a newline character will terminate each input line. You may assume that all of the input values will be logically correct (no negative or missing values, for instance), that they will be in the specified ranges, and that they will be given in the specified order. Unless indicated otherwise, you may also assume that all data values will be in the range 1 to 999.

Postal Rates Input File
The first three lines of this input file are preceded by an appropriate label and specify:

- a positive integer representing the maximum allowable weight of a package
- a positive integer representing the maximum allowable length of a package
- a positive integer representing the maximum allowable size of a package

The fourth line is blank. The fifth line contains column labels, which should be ignored. The remaining lines of this input file will each contain:

- a positive integer representing a possible package weight
- a decimal value representing the cost of shipping a package of the indicated weight

Use two one-dimensional arrays to store this table. The cost of shipping a package can then be determined by searching the Weight array and using the corresponding element in the Cost array. If the weight of a package falls between two weights in the table, the higher weight should be used to calculate the cost of shipping the package.

You may also assume that there will be no more than 25 (weight, cost) pairs.

Here is an example postal rates file:

```
Maximum Weight: 50
Maximum Length: 36
Maximum Size:   72

Weight  Cost
   2    1.00
   4    1.75
   6    2.50
   8    3.25
  10    4.00
  12    4.50
  14    5.00
  16    5.50
  18    6.00
  20    6.50
  22    7.00
  24    7.50
  26    8.00
  28    8.50
  30    9.00
  35    9.83
  40   10.66
  45   11.50
  50   12.33
```

Transactions Input File
The first line of this input file contains column labels, which should be ignored.

The remaining lines of the input file will each contain:

- a positive integer representing the package number: range is 1000 to 9999
- a positive integer representing the weight of the package
- a positive integer representing the first dimension of the package
- a positive integer representing the second dimension of the package
- a positive integer representing the third dimension of the package

There is no guaranteed limit on the number of transactions.

Here is an example transactions file:

```
Number  Weight  Dim1  Dim2  Dim3
 1001     50     36     1     1
 1002     35     12    16    35
 1003     23      7    13    20
 1004     60      1     1     1
 1005      1      6     6     2
 1006     18     10    10    12
 1007     51     27    49    15
 1008     49     31    10    10
 1009     41      9    36     9
 1010     52      9    10    36
 1011     39     38     8     9
 1012      6      3    17    32
```

What to Calculate:

Your program must output a table showing all of the transactions read from the input file. The table format is shown in the output section below. In particular, for each package, you must determine if it is too big, too heavy, too long, or any combination of these and indicate your conclusions as shown in the example output file below. If the package can be mailed, you must determine the cost of mailing it.

Output description and sample:

Your program must write its output data to a file named `package.out` — use of any other output file name **will** result in a score of zero. A sample output file produced from the sample input files above is shown below:

```
Programmer:  Donald Allison
Parcel Post

Number   Weight   Cost   Too Big   Too Heavy   Too Long
-------------------------------------------------------
 1001       50   12.33
 1002       35   *****       x
 1003       23    7.50
 1004       60   *****                  x
 1005        1    1.00
 1006       18    6.00
 1007       51   *****       x          x           x
 1008       49   12.33
 1009       41   11.50
 1010       52   *****       x          x
 1011       39   *****                             x
 1012        6    2.50
-------------------------------------------------------

Packages processed:  12
Packages rejected:    5
```

As usual, the first line of your output should identify you by name, as shown. The second line should include the title "Parcel Post" only. The third line should be blank. The fourth and fifth lines should contain the specified column labels and a row of delimiters to mark the top of the table.

Next your output file will contain a table, with a line of output for each package. Each line should contain the package number and the package weight. If none of the criteria for a package are exceeded, print the cost of shipping the package; otherwise print `"******"` in the Cost column and print a lower-case `'x'` in the appropriate column(s) to indicate which of the criteria are exceeded.

After the last line of the table, print a line of delimiters and then display the total number of packages processed and the number of packages that exceeded any of the criteria.

You are not required to use the exact horizontal spacing shown in the example above, but your output must satisfy the following requirements:

- You must use the specified header and column labels, and print a row of delimiters before and after the table body, as shown.
- You must arrange your output in neatly aligned columns. Use spaces, not tabs to align your output.
- You must use the same ordering of the columns as shown here, and print the Cost with precision two.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code should be followed.

Documentation:

- You must include the honor pledge in your program header comment.
- Your header comment must describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.
- Precede every function you write with a header comment. This should explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), and state reasonable pre- and post-conditions.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Coding:

- Use named constants instead of variables where appropriate.
- Use `double` variables for all cost values, and `int` variables for all other input data.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement **other than** a constant or variable declaration, function prototype or comment. Blank lines do not count.
- You must write at least four functions, besides `main()`. For reference, my solution uses nine: one for input, four for output, three to determine if the package should be rejected, and one to determine the mailing cost.
- You must write at least one `void` function, at least one `bool` function, at least one function of type `int`, and at least one function of type `double`.
- The definition of `main()` must be the first function definition in your source file. You may use file-scoped function prototypes and you may use file-scoped constants.
- You may not use file-scoped variables of any kind.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed.

Your submission that receives the highest score will be graded for adherence to these requirements, whether it is your last submission or not. If two or more of your submissions are tied for highest, the earliest of those will be graded. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later.

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen.

At minimum, you should test your program on **all** the posted input/output examples given along with this specification. The same program that will be used to test your solution generated those input/output examples. You could make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Pledge:

Each of your submissions to the NAG must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
//    On my honor:
//
//    - I have not discussed the C++ language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C++ language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C++ language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    - I have not designed this program in such a way as to defeat or
//      interfere with the normal operation of the Automated Grader.
```

Failure to include this pledge in a submission is a violation of the Honor Code.