Willis Towers Watson

Developer Problem

1/ Purpose and Instructions

Purpose

As it is difficult to fully assess somebody's abilities at an interview, particularly their programming skills, we give a small programming task to all potential recruits. The problem is a fairly simple one, which should be completed in the applicant's own time.

We are not concerned solely with whether or not the program works, but also with how tidy, well-structured, robust and adaptable it is. **This is a very important part of our hiring process and our assessment criteria cover the various dimensions of the software development lifecycle. Therefore we recommend that candidates give this adequate consideration and address this task as they would do for any other professional assignments in their current workplace.**

Instructions

- **Programming language:** our preferred language is C#, but if you wish to answer the evaluation question in another language, then please feel free to do so.
- **Packaging your submission**: please submit your solution in the form of an archive file (e.g. zip), removing all binaries such as .exe or .dll to ensure successful transmission by email. It is therefore essential that your solution is provided with all required source and scripts for a Willis Towers Watson engineer to be able to rebuild and execute the relevant programmes as part of our evaluation process.
- **Time or other constraints**: we would much prefer that you submit a well-designed functional software implementation, but if you are pressed for time it would be better to submit a specification which describes how you would tackle the problem than a hurried solution that does not address any of the issues. Similarly, a specification would be perfectly acceptable if you have no access to a suitable programming environment.

2/ Task

Background

One of the major jobs that we do is claims reserving, which involves assessing how much money is likely to be paid in future in respect of the claims that arise from a set of insurance policies that have already been taken out. The statistical analysis of this is based around triangles of payment figures for previous claims.

A very simple example triangle might look like this (the triangle is the shaded bit):

Incremental Claims Data		Development Year		
		1	2	3
Origin Year	1995	100	50	200
	1996	80	40	?
	1997	120	?	?

The numbers in the triangle are the amounts of payments made in respect of claims on a particular block of insurance policies.

The block of claims will be broken into origin years. For example, all claims originally happening in 1996 will go into the 1996 row of the triangle.

For each origin year grouping of claims we will see development of payments over time as settlements are made. In the example above we see that payments of 100 were made in respect or origin year 1995 during the year 1995, 50 during the year 1996 (development year 2) and 200 during 1997 (development year 3). At the end of 1997 the total of claim payments in respect of origin year 1995 is 350 = 100 + 50 + 200.

The purpose of the triangle is to try to estimate what numbers should go in the empty boxes (the ones with question marks in them) by looking at the patterns in the triangle – the numbers in the empty boxes are the amounts we expect to have to pay in the future in respect of the claims that occurred in 1996 and 1997.

To do this in practice, it is easier to spot patterns if we 'accumulate' the data in the triangle, so that rather than showing the amount paid in any particular development year, it shows the total amount paid in all development years up to and including the one we're looking at.

The accumulated version of the above triangle is:

Cumulative Claims Data		Development Year		
		1	2	3
Origin Year	1995	100	150 (=100+50)	350 (=150+200)
	1996	80	120 (=80+40)	?
	1997	120	?	?

The Problem

You are required to create a program to read in incremental payment data from a comma-separated text file, to accumulate the data and to output the results to a different comma-separated text file. This process represents the creation of the cumulative data triangle from the incremental data triangle above.

Example Input Data

A short input file might contain the following:

```
Product, Origin Year, Development Year, Incremental Value
Comp, 1992, 1992, 110.0
Comp, 1992, 1993, 170.0
Comp, 1993, 1993, 200.0
Non-Comp, 1990, 1990, 45.2
Non-Comp, 1990, 1991, 64.8
Non-Comp, 1990, 1993, 37.0
Non-Comp, 1991, 1991, 50.0
Non-Comp, 1991, 1992, 75.0
Non-Comp, 1991, 1993, 25.0
Non-Comp, 1992, 1992, 55.0
Non-Comp, 1992, 1993, 85.0
Non-Comp, 1993, 1993, 100.0
```

This example file contains *two* triangles – one for a product called 'Comp' and one for a product called 'Non-Comp'. The first row contains column headings, and the subsequent rows contain the data, so, for example, for accidents occurring on the Non-Comp product in 1990, 45.2 was paid in 1990, 64.8 was paid in 1991 and 37 was paid in 1993.

The output file corresponding to the above input file would be:

```
1990, 4
Comp, 0, 0, 0, 0, 0, 0, 0, 110, 280, 200
Non-Comp, 45.2, 110, 110, 147, 50, 125, 150, 55, 140, 100
```

The first line gives the earliest origin year (i.e. 1990) and the number of development years (in this case ranging from 1990 through to 1993 i.e. 4).

After the first line, there is a line for each triangle. The first field in the line gives the name of the product. The subsequent fields are the accumulated triangle values.

General Considerations

The data may well contain more than two triangles and there may be many more than four origin years.

The example input data file given is very 'clean'. In practice data files may contain errors or the data may be in an unexpected format.

Incremental values in the input data may have been left out of the input file if they are zero. Origin years with no claims may have been left out of the input file.

You should consider these issues and other potential problems with the input file, and how your program might deal with them.

The majority of solutions that we see do not work first time, hence you will need to bring the source code along to be able to change it and recompile it.