

Boolean Expressions and Selection

This programming assignment uses many of the ideas presented in section 5 of the course notes, as well as material from sections 3 and 4 that you may have used previously. You are advised to read those sections carefully. Read and follow the following program specification carefully. You should be able to use the same data processing approach from the previous two programming assignments

Your score on this project will be a weighted average of the score you receive for runtime testing and the score you receive for following the instructions in the Programming Standards section below.

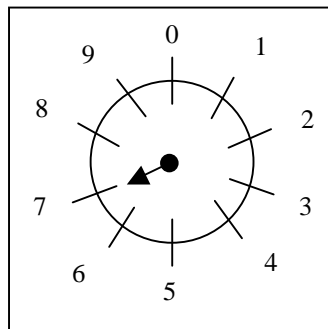
The Program Specification:

Gas Bill

This program is modified from a similar problem published in *Pascal Programming and Problem Solving with Software, 4/e*, Sanford Leestma and Larry Nyhoff, © 1993, Prentice Hall Engineering/Science/Mathematics Division.

The Good Glow Gas Company (GGGCo) bills customers for gas usage by the cubic meter. The company maintains a gas meter at every customer's location, and once a month a meter reader makes the rounds to each customer's location and reads the current value on the meter. The company then uses the previous month's reading and the current month's reading to determine how much gas the customer used that month.

The gas meter is a circular device that has a tick mark around the outer edge for every number from 0 to 9999. A needle points to a tick mark on the meter, and moves to the next tick mark each time a cubic meter of gas is used. When the needle points at mark 9999 and a cubic meter of gas is used, the meter is reset to 0. Below is a schematic diagram of a meter, with tick marks displayed only for each thousand cubic meters. In the schematic, the needle is currently pointing at 7, meaning the current meter reading is 7000.



The meter reader simply looks at the meter, and records the value of the tick mark at which the needle is currently pointing to get the current reading. The reading from the previous month is stored in GGGCo's computer system and accessed from there to compute the total amount of gas used. The total amount used in the month is the number of tick marks that the needle crossed from the previous reading to the current reading, not including the previous reading tick mark but including the current tick mark. So in the meter shown above, if the previous month's reading had been 4000 (needle pointing at the 4), the total amount of gas used this month would be 3000 cubic meters.

Once GGGCo has determined how much gas a customer has used during the month, they compute the amount of money the customer owes for that gas usage. For any amount of gas up to and including 300 cubic meters, the company charges a fixed rate of \$21.00. For any amount over 300 cubic meters and less than or equal to 600 cubic meters, they charge an additional \$0.06 per cubic meter used. For any amount over 600 cubic meters and less than or equal to 800 cubic meters, they charge \$0.04 per cubic meter. Finally, for any amount over 800 cubic meters, they charge \$0.025 per cubic meter. For example, if the total amount of gas used by a certain customer were 470 cubic meters, their gas bill would be computed as follows:

$$\$21.00 + 170 * \$0.06 = \$31.20$$

Here, the \$21.00 covers the first 300 cubic meters, and the remaining 170 cubic meters are charged at \$0.06 per cubic meter.

First write an algorithm in outline form, using the modular decomposition method, to solve the following problem: Read from each line of an input file two integer values — the meter reading of the previous month for some customer, and the meter reading for the current month for the same customer — and compute the total amount of gas used during the month by that customer, the gas bill for the month (in dollars), and the average price per cubic meter of gas used in the month (in dollars). These three values, as well as the two meter readings, are to be output in a labeled tabular format as described below in the output file section. Also, compute the total amount of gas used by all customers listed in the file, the total amount of money that will be collected from all of these customers, and the overall average price per cubic meter of gas for all the customers in the file. Second, use this algorithm to write a C++ program that solves the problem. The algorithm should be included at the top of the source code as comments following the honor code pledge.

Note on gas bill computation:

You will be printing out two places after the decimal for the dollar amounts of the gas bill and the average price per cubic meter of gas. Due to the way floating point numbers are represented in the computer, the computation may lead to a situation where the internal representation is slightly different than the actual value if computed with a calculator. This can make the rounding in setprecision produce dissimilar results. Furthermore, if this value is used as you compute the average price and the total gas bill for all customers, the roundoff error may carry over to those computations.

In order to avoid these problems with rounding, you should explicitly perform the rounding to two decimal places. The rounding should occur immediately **after** you have computed the gas bill, and **before** you use the gas bill amount in any other computation. The code to round the variable `gasBill` to two decimal places is

```
// Round the gas bill to two digits
gasBill = floor (gasBill * 100 + 0.5) / 100.0 ;
```

The floor function returns the nearest integer that is not greater than the input. For example, `floor(9.2)` is equal to 9. Also, `floor(9.8)` is equal to 9. The floor function is in the math library, so you need to include `math.h` to use this function. To see that this rounds properly, consider the following examples. First, if `gasBill` is 58.975, then the value that gets input to the floor function is $5897.5 + 0.5 = 5898.0$. The floor of this number is 5898, and when the final division is done, the result is 58.98. If `gasBill` is 144.27, then the input to the floor function will be 14427.5. The floor of this number is 14427, and when the final division is done, the final value is 144.27.

Input file description and sample:

Your program **must** read its input from a file named `gasusage.dat` — use of another input file name will result in massive deductions. The first line of the input file contains a label, which should be ignored. Each remaining line of the input file will contain two integers. The first integer is the meter reading from the previous month, and the second integer is the meter reading from the current month.

You may assume that all the input values will be logically correct (only values from 0 to 9999, inclusive). For instance:

Previous Reading	Current Reading
200	450
575	1250
2620	4250
1710	5730
8725	150
6180	6290

Note that you must **not make any assumptions** about the number of lines of data in the input file. Your program must be written so that it will detect when it is out of input and terminate correctly. You have used this technique in your previous programs; see also slides 4.20 – 4.22 in the course notes.

Output description and sample:

The first line of your output should include your name only; the second line should include the title “Gas Bills” only; the third line should be blank; the fourth line should display the column labels shown below; the fifth line should consist of some delimiting symbol to separate the column labels from the body of the table.

Next your output file will contain a table, with one line of output for each line of numeric data in the input file. Each line of the table should list the previous meter reading, the current meter reading, the amount of gas used that month, the bill for the gas used (in dollars), and the average price per cubic meter (in dollars) of the gas used that month by the customer. There should be a line of delimiters immediately after the last line of the table body. The final line of the output contains the label "Totals:" and the total amount of gas used by all customers in the file, the total amount of the gas bills for all the customers in the file (in dollars), and the overall average price per cubic meter (in dollars) for all the customers in the file. These three values should be lined up in the appropriate columns of the main table, as shown below.

The values for all dollar-valued data (the bill and the average) should have precision 2, i.e., show two digits after the decimal. The values for all integral data (meter readings and amount of gas used) should be displayed without decimal points.

Your program must write output data to a file named `gasbill.dat` — use of any other output file name will result in a massive deduction of points. The sample output file shown below corresponds to the input data given above:

Pamela J Vermeer				
Gas Bills				
Previous	Current	Amount Used	Gas Bill(\$)	Average \$/meter ³
200	450	250	21.00	0.08
575	1250	675	42.00	0.06
2620	4250	1630	67.75	0.04
1710	5730	4020	127.50	0.03
8725	150	1425	62.63	0.04
6180	6290	110	21.00	0.19
Totals:		8110	341.88	0.04

You are not required to use this exact horizontal spacing, but your output must satisfy the following requirements:

- You must use variables of type `int` for the meter readings and the total amount of gas used.
- You must use variables of type `double` for all the real number variables, your choice.
- All decimal values must be printed to show the same number of decimal places as in this sample.
- You must use the specified header and column labels, and include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. Note that while the Curator doesn't deduct points for horizontal alignment, the TA who may grade your source code for programming standards may do so.
- The total values on the last line must be lined up in the corresponding column of the main table.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. A copy of *Elements of Programming Style* is included with the course notes — if you don't have a copy it is strongly suggested you read the on-line edition (available from the course web page).

Some specifics:

- You must include header comments specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- 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.
- You must use named constants for values that are constants in the program.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Your submission that receives the highest score may 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.

Incremental Development:

You'll find that it's easier and faster to produce a working program by practicing incremental development. In other words, don't try to solve the entire problem at once. First, develop your design. Remember that a careful analysis of the program and development of an algorithm on paper is an extremely important part of the problem-solving method. Also, the algorithm developed in this process is the modular decomposition required in the comment section, as discussed above.

When the time comes to implement your design, do it piece by piece. Here's a suggested implementation strategy for this project:

- First, write the code necessary to read the entire input file. This should be similar to the input code used in your previous programs. To test your work, include code to write what you're reading (and nothing else) to the output file. There's not much point in worrying about processing the data further until you know you're reading it correctly.
- Second, add the code to compute the total amount of gas used for each pair of input meter readings, and print that to the output file. Also print the specified header to the output file. Verify that the results are correct; if not, determine why and fix the problem.
- Third, add the code to compute the gas bill for each line and print that to the output file. Verify that the results are correct; if not, determine why and fix the problem.
- Fourth, add the code to compute the average price per cubic meter for each line of data and print that to the output file. Verify that the results are correct; if not, determine why and fix the problem.
- Fifth, add the code to accumulate the total amount of the bills in the file, the total amount of gas used by the bills in the file, and the overall average price per cubic meter. Output that to the file. Verify that the results are correct; if not, determine why and fix the problem.

Now you have a substantially complete program. At this point, you should clean up your code, eliminating any unnecessary instructions and fine-tuning the documentation you already wrote. Check your implementation and output again to be sure that you've followed all the specifications given for this project, especially those in the Programming Standards section above. Be sure you have entered the Honor Code statement at the beginning of your program, and your modular decomposition outline. At this point, you're ready to submit your solution to the Grader.

Testing:

At minimum, 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. You should make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be. Be sure your tests exercise all of the code in your program!

Pledge:

As with the previous projects, each of your submissions to the Curator 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.