

Reading to Input Failure, Simple File I/O, Arithmetic Calculations, Using Standard Functions

This programming assignment uses many of the ideas presented in sections 3 and 4 of the course notes, so you are advised to read those carefully. Read and follow the following program specification carefully. This program is considerably simpler than the payroll program from Project 1, but don't underestimate it, especially if you've not done much programming before.

This project will be worth 6% of your final grade. You will receive two scores on this project. You will receive a score for runtime testing from the EAGS (90% of the total program grade). You will also receive a software engineering score for following the instructions in the Programming Standards section below (worth 10% of the total program grade).

The Program Specification:

Newton's Law of Cooling

Suppose an object is heated and then placed in a container filled with some cooler medium (like air, water, oil, etc). Obviously, the temperature of the object will gradually decrease, approaching the temperature of the surrounding medium. For this assignment you will write a simple program that will illustrate this cooling effect by using one of the standard mathematical models that apply to this situation.

Assume that the temperature of the surrounding medium does not change as the object cools. That's a reasonable assumption if the medium is circulated and has a relatively large volume. In that case, it is possible to show that the temperature of the object after t seconds is modeled by the formula

$$T = \alpha + (T_0 - \alpha) \times e^{-k \times t}$$

where:

- T_0 is the initial temperature of the object (when it is placed in the medium)
- α is the constant temperature of the surrounding medium
- t is the number of seconds since the object was placed in the medium
- k is a constant that is determined by the thermal properties of the object and of the surrounding medium
- e is the Euler number, approximately 2.71828

The rate at which the object is cooling at any instant is also of some interest. It can be shown that the rate of cooling is given by the formula:

$$R = k \times (T_0 - \alpha) \times e^{-k \times t}$$

For example, suppose that a block of steel is heated to 800° and then placed in a vat of oil at a temperature of 100°. Suppose that the constant k equals 0.05 in this case. Then after 60 seconds the temperature of the block of steel would be

$$T = 100 + (800 - 100) \times e^{-0.05 \times 60} = 100 + 700 \times e^{-3} \approx 100 + 700 \times 0.0497871 \approx 134.85$$

and the block of steel is cooling at a rate of

$$R = 0.05 \times (800 - 100) \times e^{-0.05 \times 60} = 35 \times e^{-3} \approx 35 \times 0.0497871 \approx 1.74$$

It's not necessary, or even useful to understand the derivation of the formulas given above to complete this assignment.

Input file description and sample:

Your program **must** read its input from a file named `cooling.dat` — use of another input file name will result in a score of zero. The first line of the input file contains column labels which should be ignored. The second line contains the initial temperature of the object, followed by the constant temperature of the medium. The third line contains a column label which should be ignored. Each remaining line of the input file will contain a single time value, followed by a character to indicate the units of the measurement - 's' or 'S' for seconds, and 'm' or 'M' for minutes

You may assume that all the input values will be logically correct (no negatives, for instance). For instance:

Object	Medium
800.00	100.00
Time	
10.0 s	
20.0 S	
30.0 s	
40.0 S	
50.0 s	
60.0 S	
1.0 M	
2.0 m	

The input values won't always have the nice patterns shown in this example so don't make any unjustified assumptions. Note that you must **also 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's out of input and terminate correctly. A technique for this will be discussed in class and was used in the payroll program from Project 1; see also slides 4.20 – 4.21 in the course notes.

What to Calculate:

You will write a program to will read the input file and use the given values to compute:

- the temperature of the object at the given time
- the rate at which the object is cooling at that time

To perform these calculations, you will need to use the formulas given above. Assume the thermal constant, k , is always 0.05. The formula requires using the natural exponential function, to calculate the power of e . Fortunately, this may be found among the standard library functions in C++. The natural exponential function is called `exp()`; it takes one parameter of type `double` and returns the square root of that parameter. So, the statement:

```
double aPower = exp(3.14);
```

would assign (approximately) the value 23.1039 to the variable `aPower`. You must use a compiler directive to include the standard header file `cmath` in order to use this function.

Additionally, because the formula requires the value in seconds, it may be necessary to convert from minutes into seconds by multiplying the value by 60. Remember, the character can be either upper or lower case, so you need to be able to deal with both.

Output description and sample:

A sample output file produced from the sample input file above is shown below. The first line of your output should identify you by name, as shown. The second line should include the title “Newton’s Law of Cooling” only. The third line should be blank. The fourth and fifth lines should display the initial temperature of the object and of the surrounding medium, labeled exactly as shown.

Next your output file will contain a table, with one line of output for each time value given in the input file. Each line of the table should list the time (in seconds, regardless of how it was in the input file), the object’s temperature at that time, and the rate at which the object is cooling at that time. The table columns should have labels, exactly as shown. There should be a line of delimiters immediately after the column labels, and another to mark the end of the table. Each line of output (including the last) should be followed by a newline character.

The time values should be printed with precision 1, and the temperature and rate values should be printed with precision 2, as shown.

Your program must write its output data to a file named `temp.out` — use of any other output file name will result in a score of zero.

```
Programmer:  Chris Knestrick
Newton's Law of Cooling

Initial temp of object:  800.00
Temperature of medium:  100.00

  Time      Temp      Rate
-----
 10.0     524.57     21.23
 20.0     357.52     12.88
 30.0     256.19      7.81
 40.0     194.73      4.74
 50.0     157.46      2.87
 60.0     134.85      1.74
 60.0     134.85      1.74
120.0     101.74      0.09
-----
```

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

- You must use variables of type `double` for all the real numbers. If you use `float` variables, your answers may not be as accurate as needed.
- 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. Use spaces, not tabs to align your output. Note that while the EAGS doesn’t deduct points for horizontal alignment, the TA who grades your source code for programming standards may do so.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line, including the line of hyphens marking the end of the table.

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 I strongly suggest 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.
- Your header must include a design outline of your program. See slide 3.2 for details.
- 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.
- Use named constants instead of variables where appropriate.
- Use of boolean operators (AND, OR, NOT) instead of nested if's where appropriate to make code more readable (see slide 5.15 for an example).
- 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. The payroll code from Project 1 is a good guide.

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. 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 somewhat similar to the input code used in the payroll program. 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 temperature of the object and change your output code to print that to the output file, after printing the time. Also print the specified header to the output file, and get the precision settings right. Verify that the results are correct; if not, determine why and fix the problem.
- Third, add the code to compute the rate of cooling and print that to the output file. Verify that the results are correct; if not, determine why and fix the problem.
- Fourth, clean up your output format and be sure it matches the specification above. 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. At this point, you're ready to test your program on all the posted input/output examples before submitting your solution to the EAGS.

Hints:

This program requires that you know how to manage file-oriented input/output operations — the slides and the text provide good examples and guidelines. Your program must read lines of input data until there is no more data to be processed. You may want to use the same logical design as in the payroll program from Program 1.

You'll have to use **manipulators** to manage the formatting of your output. (Use of `printf` is strictly forbidden, you must use C++ streams for I/O in the programs for this course.) Read the discussion on slides 4.16 – 4.19 carefully, there are important clues there. The payroll program source that was provided for Project 1 also shows how this is done.

You also have to do some mathematical calculations that go beyond mere arithmetic. The C++ language includes most of the standard mathematical functions, including a trigonometric, logarithm and exponential functions. To use them, you need to add another `#include` at the beginning of your program: `#include <cmath>`

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. Those were generated by the same program that will be used to test your solution. 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 EAGS 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.