ECOR 1606 Lab #9

Note: To prove your attendance at this lab, submit file "*lab9.cpp*" containing your program by the end of the lab period.

Introduction:

This laboratory illustrates a "bottom-up" approach to problem solving. You will start by writing some functions, using some supplied functions, and will write the main function last. This differs from a "top down" approach in which the main function is written first (as in the gas company example in the notes).

Background:

As a liquid moves through a horizontal pipe, the pressure drops due to friction between the liquid and the walls of the pipe. In order to calculate the pressure drop, it is necessary to first calculate the *friction factor, f,* and the *Reynolds number, R.* Ultimately, you are to write a program that reads in the flow velocity and the distance between two points in a pipeline, and prints a table of the pressure drop for pipes of different diameters.

Part I:

Start Dev-C++, open the *framework.cpp* file and save it as *lab9.cpp* (on the desktop).

You are going to start by including function *calcFrictionFactor* that has been written for you. You can find the code in *calcFrictionFactor.txt* in the *lab9* folder.

Put this function just above the *main* program (i.e. between the *sqr* function and the *main* function).

Below is an explanation of how the friction factor is calculated by this function. It is recommended that you read it. However, if you do not completely understand the description, it will not prevent you from completing the rest of the lab.

The friction factor for the flow through a pipe depends upon the Reynolds number for the flow, the diameter of the pipe, and the roughness of the inside of the pipe. An equation for the friction factor, *f,* is:

$$f = \left[2\log_{10}\left(\frac{\varepsilon/D}{3.7} + \frac{2.51}{R\sqrt{f}} \right) \right]^{-2}$$

where R is the Reynolds number for the flow through the pipe (dimensionless),
D is the diameter of the pipe (in metres),
ε is the roughness of the inside of the pipe (also in metres).

Unfortunately, because f is on both the left and right hand side of the equation, the equation cannot be used directly to determine f. However, in this case, we can estimate f by employing a strategy of s*uccessive approximation*. If we guess at the value of f and plug this value into the right hand side of the equation, the result will be a better value for f. And, if we then plug this better value back

into the right hand side, the result will be a still better value, and so on. A reasonable initial guess can be generated by applying the Blasius formula for smooth pipes ($f = 0.3164\ R^{-0.25}$).

Example:

Assume R = 10,000, D = 0.1m, and ε = 2.6 mm

Initial guess for $f = 0.3164\ R^{-0.25} = 0.03164$
Plugging 0.03164 into the right hand side of the formula gives f = 0.0581334
Plugging 0.0581334 into the right hand side of the formula gives f = 0.0570566
Plugging 0.0570566 into the right hand side of the formula gives f = 0.0570852
Plugging 0.0570852 into the right hand side of the formula gives f = 0.0570845
… and so on

The longer we go, the better the answer will become, but the law of diminishing returns applies, and it takes only a few iterations to generate reasonable values. For this lab, stop the process when the difference between the value plugged into the right hand side of the equation and the calculated value for f is less than or equal to 0.000001. At this point the result will be accurate to five decimal places.

Part II:

Next you are going to write function *calcReynolds*:

calcReynolds Function:
The Reynolds number for flow through a pipe is given by

$$R = \frac{\rho V D}{\mu}$$

where ρ is the density of the fluid (in kilograms per cubic metre),
V is the velocity of the fluid (in metres per second),
D is the diameter of the pipe (in metres), and
μ is the viscosity of the fluid (in Newton seconds per square metre).

The units all cancel, and R is dimensionless. *calcReynolds* is given ρ, V, D, and μ (all in the units given above) and computes and returns R. The function prototype is:

```
double calcReynolds(double rho, double V, double D, double mu);
```

Write this function below *calcFrictionFactor* in your program. Remember that to convert the function prototype to a function, you remove the ";" at the end and add a "{" and then write the body of the function, ending with "}".

Part III:

Now we need to test our *calcReynolds* function. In your main program, write a sentinel loop to to read in values for rho, V, D, and mu (until 0 0 0 0 is entered). After getting the values, call the function and output the result. Check the answers with *part3.exe* to make sure they are correct.

Part IV:

Now you are going to write function *calcPressureLoss*:

calcPressureLoss Function:
As a fluid flows along a pipeline (assumed to be horizontal), the pressure drops. The pressure loss, Δ, between two points can be calculated using:

$$\Delta = f \frac{L}{D} \rho \frac{V^2}{2}$$

where f is the friction factor for the flow (dimensionless)
L is the distance between the two points (in metres)
D is the diameter of the pipeline (in metres)
ρ is the density of the fluid (in kilograms per cubic metre)
V is the velocity of the fluid (in metres per second).

Assuming the units given, the calculated pressure loss will be in Pascals. *calcPressureLoss* is given V, *L*, *D*, ρ (all in the units given above), the viscosity of the fluid (in Newton seconds per square metre), and the roughness of the pipe (in metres), and computes and returns the pressure loss (in Pascals). This function uses the two functions above (i.e. *calcReynolds* and *calcFrictionFactor*). The function will return -1 (meaning that no answer can be computed) if the Reynolds number for the flow is less than 3500 (because the formula used for computing the friction factor is not valid at low Reynolds numbers). The function header is:

```
double calcPressureLoss(double V, double L, double D, double rho, double mu, double epsilon);
```

Write this function below *calcReynolds* in your program. Remember that to convert the function prototype to a function, you remove the ";" at the end and add a "{" and then write the body of the function, ending with "}".

Part V:

Now we need to test our *calcPressureLoss* function. In your main program, comment out the code you wrote to test *calcReynolds* (don't delete it, in case we need it again), and write another sentinel loop that accepts V, L, D, rho, mu and epsilon (0 0 0 0 0 0 to exit). After getting the input values, call the function and output the result. Again, check that your answers are correct (see *part5.exe*). (Note that you may assume that both calcReynolds and calcFrictionFactor work correctly at this point.) Does it matter if we give the variables different names in the main program from those used in *calcPressureLoss*? Try it and see.

Part VI:

Write a **function** that generates a table showing the pressure drops for pipeline diameters from 0.05m to 0.75m (in steps of 0.05m). The inputs to this function are the velocity of the fluid (*V*), the distance between two points on the pipeline (*L*), the density of the fluid (ρ), the viscosity of the fluid (μ), and the roughness of the inside of the pipe (ε). The function header must be:

```
void printTable(double V, double L, double rho, double mu, double epsilon);
```

This function should use *calcPressureLoss* (above). Have your function output stars in place of a value if a pressure drop cannot be computed (i.e. if *calcPressureLoss* returns -1).

Use a "for" loop to output your table, and make sure that you use the required output formatting commands to make your table look exactly like the one below. Note that you should **always** use integers (not doubles) in your "for" loops, due to rounding errors with doubles. Thus the following is a bad idea:

```
double x;
for (x = 0.5; x<=0.75; x+=0.05) { // you may get 14 or 16 lines in your table instead of 15
   ...
}
```

instead you want to use an integer and convert to double inside the loop:

```
double x;
int i;
for (i=0; i<15; i++) { // this way we are sure to get 15 lines in our table
   x = (i+1)*0.05;
   ...
}
```

```
  Diameter (m)     Pressure Drop (Pa)
-------------------------------------------
        0.05                 15691.5
        0.10                  6638.2
        0.15                  4036.7
        0.20                  2843.1
        0.25                  2168.9
        0.30                  1740.0
        0.35                  1445.0
        0.40                  1230.7
        0.45                  1068.5
        0.50                   941.8
        0.55                   840.3
        0.60                   757.4
        0.65                   688.4
        0.70                   630.3
        0.75                   580.6
```

Part VII:

Comment out the main program code that you wrote in part V. (Again, do not delete it in case we need it later.)

Write a main function (to test Part VI and complete your program) that reads in the flow velocity for a pipeline (in metres per second) and the distance between two points on the pipeline (in meters), and produces a table showing the pressure drops for pipeline diameters from 0.05m to 0.75m (in steps of 0.05m). The sample executable, *lab9soln.exe*, should give the idea. Your program should loop, repeatedly reading in values and producing values, until -1 -1 is entered. Some basic error checking is required. Both the flow velocity and the distance must be greater than or equal to zero. If this is not the case, your program should output an error message and ignore the invalid data. Be sure to try the inputs: 0.01 and 100 to be sure that your table produces asterisks properly.

Assume that the pipeline contains water at 20 degrees C (ρ = 998.2 kg/m^3, μ = 1.002x10^{-3} N-s/m^2) and that the pipe is made of commercial steel (roughness, ε = 0.045mm). Declare these required constants within your main function. Making them global is inappropriate because the other functions should not directly access these values. The other functions are given all of the values they need as parameters. Note that the roughness in given in mm. What units does *calcFrictionFactor* require? (Hint: A conversion is required in the **main** function. Do **not** change *calcFrictionFactor*!)