ENGR 151

Fall 2023

Lab 5: Getting Started with C++

The purpose of this lab is to orient you on how to set up a programming environment for C++. There are PDF documents posted on Canvas to walk you through the process. Make sure to ***read carefully!!*** There are also two exercises to familiarize you with C++ language and syntax. **Note: Please go through the setup guide carefully before proceeding with this section of the lab.**

Exercise 1: Compiling, Debugging, and Running a C++ program

Compilation

"Compiling" a C++ program is the process of converting the "source code" (English) to "machine language" (0s and 1s) which a computer can understand and execute. We use a program called a "compiler" to compile the written C++ program. The compiler that we'll use in this course is called g++ which is an open source compiler distributed by GNU (look it up if interested).

Many of you are proficient in using GUI (graphical user interface) tools for programming and generally interacting with a computer. However, in this course we'll use CLI (command line interface) tools like g++. Most CLI tools take input arguments on the command line. The way that we'll use g++ is as follows

g++ <compilation options> <source_code> -o <output_executable>

This format allows us to specify all the relevant compilation options. Sometimes, the default compilation options are enough. Sometimes we need additional options to ensure that we get what we want. The general compilation command that we'll use for labs here will be

g++ -Wall -Werror <filename>.cpp -o <filename>.out

-Wall : "Warn all". This flag turns on the g++ feature that reports all warnings irrespective of whether the compiler thinks they are fatal or not.

-Werror : This flag tells the compiler to treat all warnings as errors. *The core difference between a warning and an error is that a warning is a probable error in the program that doesn't halt the compilation process. An error is a fatal mistake in the program because of which the compiler cannot proceed further.* Enabling this flag ensures that you catch a lot of probable mistakes early on and fix them.

-o : The output flag. While the order of the flags described above doesn't change the meaning of the compilation, this flag is a little peculiar. The argument succeeding this flag must be the name of the output executable. If you omit **-o <output_executable>** altogether from your compilation command, g++ creates a default executable named **a.out**. The convention is to name the executable **<filename>.out** where **<filename>.cpp** is the name of the source code file.

Warning: Be careful not to accidentally put the source file <filename>.cpp after the -o instead of the .out file. (i.e. g++ run -o test.cpp)
If you do this, your cpp file's contents will be overwritten with the executable.

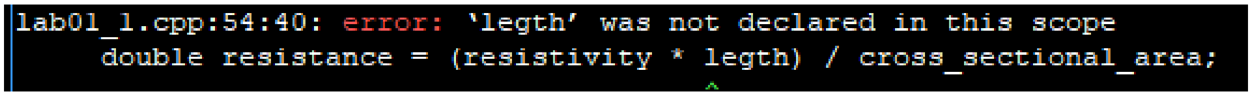
Debugging and Running a C++ Program

Now that you know how to compile a C++ program, let's put that knowledge to use. Download **calculate_resistance.cpp** from Canvas and try to compile it using the instructions given above. Call your executable **lab1_01.out.**

So what happens? There seem to be some compilation errors. That's because we have deliberately introduced a few "bugs" in the code. A mistake in the code, whether syntactic (related to the C++ syntax) or semantics (related to what the program does) is called a **bug**.

The term "bug" has a very interesting history. In the 1940s computers were made up of huge vacuum tubes (you can see a small part of the ENIAC computer in the main entrance of BBB) which needed to be programmed by hand by inserting the correct wire in the correct slot. Sometimes, bugs would get in those connections and the computer would start malfunctioning. They had to manually remove those fried bugs and hence the term "debugging". Admiral Grace Hopper, a pioneer in Computer Science who invented one of the first compilers, is often credited with creating the term "bug."

The compiler produces, or at least tries to produce, nice error messages with as much information as possible to enable the programmer to easily fix the bugs. This is one of the bugs you might encounter:



```
lab01_1.cpp:54:40: error: 'legth' was not declared in this scope
     double resistance = (resistivity * legth) / cross_sectional_area;
                                        ^
```

Try to find out the bugs in this code! The **stderr** output shown above is one bug and these will help you find the syntactic errors in the source code. Re-compile the source code and check that it compiles.
To give you a brief background, this program computes the resistance(in ohms) of a cylindrical wire given the wire's length (in m), radius (in mm), and resistivity((ohm*m)$^{-1}$).

In order to run the program, type the following command

./lab01_1.out

Notice the "./" prepended to this command. This tells the system to look for the executable file only in your current directory, rather than your PATH (your PATH contains all the places where the system can look for executable files).

You will be prompted to enter user input values. Enter the wire length, wire radius, and wire resistivity in the correct order. The order can be determined by looking at the **order of the cin prompts** in the source file **calculate_resistance.cpp.**

$$R = \frac{\rho L}{A}$$

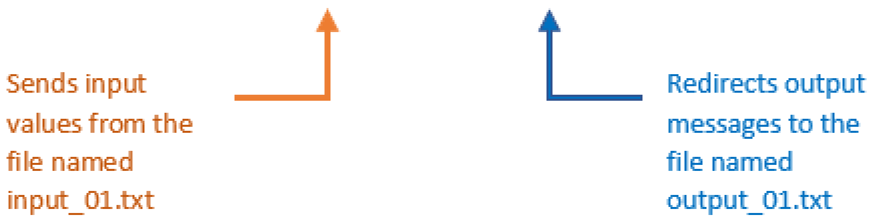
ρ = resistivity
L = length
A = cross sectional area

After you input the parameters of the wire, check if your code is calculating the resistance of the wire correctly (you can use a calculator or you can use this website http://www.endmemo.com/physics/resistance.php!).

Input Redirection Operators

You can also run your program and give it ***file input* instead of *user input***.

```
./lab01_1.out < input_01.txt > output_01.txt
```


Sends input values from the file named input_01.txt
Redirects output messages to the file named output_01.txt

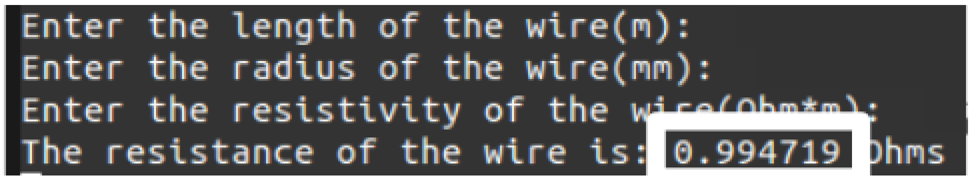
This command uses two new operators called input redirection operators

- `` `<` `` takes the file succeeding it and feeds the contents of that file as input to the program on which it is called. You can find the input file on Canvas.
- `` `>` `` takes the file succeeding it and writes the contents of the output generated by the program into the file. Note that if an output file exists, it will overwrite the contents. If the file doesn't exist, **>** will create a new file. So use this feature with caution. This is a terminal feature and it allows us to rerun the program with the same inputs over and over again without having to type out all the input values and recording the output.

Create your own input_01.txt and execute the command explained above. Then open the output file "output_01.txt". You'll notice that what used to print on the terminal is now written to a file!

In engineering, sometimes there are no autograders, correct outputs and prompts given to you. We want you to take an engineering problem and be able to check if the output you're generating is right yourself! Luckily, there is a resistance formula given above. So how do we know the output of your program is correct? First, let's create a text file with your **expected output.**

Fill in the correct value of resistance in the white box after downloading **output_01_correct.txt** from Canvas.



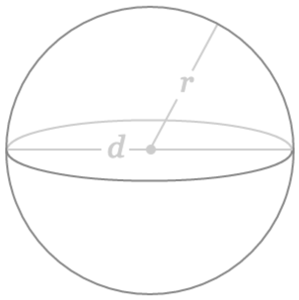
```
Enter the length of the wire(m):
Enter the radius of the wire(mm):
Enter the resistivity of the wire(Ohm*m):
The resistance of the wire is: 0.994719 Ohms
```

You can use pen and paper and a calculator or this website: http://www.endmemo.com/physics/resistance.php. Make sure that the units you input on

the website for each parameter ***match*** the units specified in the **cin** prompts. **It's important you get this right!**

(Optional) Exercise 2: Writing your own C++ program

Now that you have debugged a C++ program, let's write a new one from scratch. The problem statement is as follows



You will write a program called **sphere.cpp** that calculates the surface area, volume and ratio of volume to surface area of a sphere similar to figure above using the following relationships:

$$V = \frac{4}{3}\pi r^3$$

$$A = 4\pi r^2$$

$$R = \frac{V}{A}$$

This program should take the diameter of the sphere as input and output its surface area, volume, and ratio of volume to surface area.

Warning:
When carrying out division, be sure to use double division (4.0/3.0) and avoid integer division (4/3) in order to avoid losing precision.

For the purpose of this exercise we will define pi to be **3.14159**.

If you don't know where to start, refer to **calculate_resistance.cpp** as an example. The basic structure will remain the same but the calculations will change. Here is a table with some helpful basics

Topic	**Example**	**Notes**

//	`// your comment here`	Indicates a comment

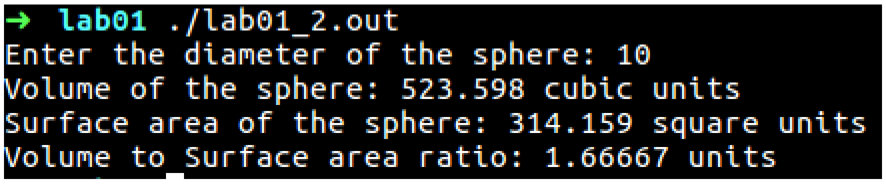
/*
*/
	`/*`
	`This is a comment block`
	`Wow wOw`
	`So cool`
	`*/`
		Creates a comment block

cout	`cout << “output message”;`	Prints output to terminal
endl	`cout << endl;`	Prints a new line
cin	`cin >> variable_name;`	Waits for user input

Declaring type of variable	`int number_of_pets;`
	`double volume;`
		In C++ you must declare a variable’s type before using it. It’s best practice to declare these at the top of the function definition or before the main function

As mentioned before, compile this program and call the output executable **lab01_2.out** (or whatever you want to call it!)

Here is a sample output after we compiled and executed our code:



```
➜  lab01 ./lab01_2.out
Enter the diameter of the sphere: 10
Volume of the sphere: 523.598 cubic units
Surface area of the sphere: 314.159 square units
Volume to Surface area ratio: 1.66667 units
```

In order to verify the correctness of your program download:

1. **instructor_output.txt**
2. **instructor_input.txt**

from Canvas and store your output into **my_output.txt** using the following command on your terminal.

```
./lab01_2.out < instructor_input.txt > my_output.txt
```

Pro tip: You can use **pow** function in **cmath** library to compute power of any number. You can use it by including the cmath library in your program. Do so by adding the following in the scaffolding part of the program along with
#include <iostream>
#include <cmath>

You can use **pow** with the following syntax

pow(number, exponent)

This will return number$^{\text{exponent}}$ as output.

Files to turn in

We're now moving to autograder.io. If you're not added yet on the ENGR151 Fall 2023 course on this website, let us know as soon as possible!
You have to submit two files to the autograder.

1. **calculate_resistance.cpp**
   Reminder: do NOT MODIFY any of the cout statements! Your output should have the same format as **output_01.txt.** We cannot assign any credit if the format is wrong.

Once you pass the setup tests, you're good to go! That means your code is compiling :)
Make sure to test your code yourself before submitting to the autograder! The results from the private tests will be released the day after the submission deadline.