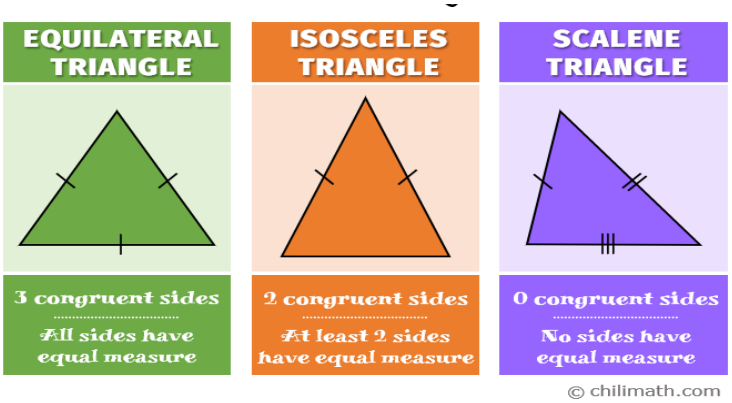
HW0: Intro to C++, Codio and HW Workflow



Problem Statement

Given 3 numbers (as potential sides of a triangle), determine if the resulting triangle is **equilateral**, **scalene**, **isosceles** or **invalid**.
Of course there is really no problem solving involved here (The next HWs do) so we jump right into implementation and **even provide you with a C++ solution** that **works (?)**

A word about C++

Since the examples in this class are written in **C++ and object-oriented style**, it's important to get familiar with that format. The **good news** is that you already know OOP from Python and you know C! C++ is **similar** in many ways, and we'll **highlight the key differences** as we go.

You do **NOT** need to write C++ code for assignments or tests — **using C is fine**. The reason we introduce this first assignment in C++ is to help you get **comfortable reading C++** code.

For **tests**, we will provide some **starter code (in C++)**. Your job will usually be to implement a **single method** (member function). **Even if the starter is in C++, you can write your solution in C and it will still work.**

Initial Implementation (in C++)

For this introductory assignment, we provide an initial solution. It is **not complete** and cannot handle all possible inputs — this is **intentional**! We want you to **take testing seriously** and **discover what is missing**.

Let's get started! First, **create the following files** in **Codio** (**inside** the `Triangle-Classifier` folder):

- `triangle.h`
- `triangle.cpp`
- `main.cpp`

Then **Simply copy/paste(Yes-only in this intro assignment[1])** the codes below into the corresponding files you just created (in Codio)**:**

triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

#include <string>
using namespace std;// see below about namespace and std

class Triangle {
private: //see comment below about private vs. public
   double a, b, c;

public: //see comment below about private vs. public
   Triangle(double side1, double side2, double side3);// this is a constructor
   string getClass(); // e.g., "equilateral", "isosceles", "scalene"
};

#endif
```

triangle.cpp

```
#include "triangle.h"
using namespace std;

Triangle::Triangle(double side1, double side2, double side3) {
   a = side1; b = side2; c = side3;
}

//the following code is not complete/correct. You need to find the problems and fix it
string Triangle::getClass() {// possible return values: "equilateral","isosceles","scalene","invalid"
   if (a <0 || b < 0 || c < 0) return "invalid";
   if (a == b && b == c) return "equilateral";
   if (a == b || b == c || a == c) return "isosceles";
   return "scalene";
}
```

[1]It is totally fine to copy/paste these codes only in this first assignment since the purpose of this assignment is to get you familiar with C++, HWs setup/ workflow, importance of checking your code, thinking about test cases and at the end, testing it with the autograder (7 times max). Copy/Pasting **is NOT allowed in the future HWs as stated in the course policy (on the syllabus and intro slides)**.

main.cpp

```
#include <iostream>
#include <fstream>
#include "triangle.h"

using namespace std;

int main(int argc, char** argv)
{
   if (argc < 3) // must provide two arguments as input
   {
       cerr << "Usage: ./main <INPUT FILE> <OUTPUT FILE>" << endl;
       return 1;  // non-zero return means error
   }
   ifstream infile(argv[1]);  // open input file
   ofstream outfile(argv[2]); // open output file

   double x, y, z;

   while (infile >> x >> y >> z) {
       Triangle t(x, y, z);
       outfile << t.getClass() << "\n" << flush;
   }
   infile.close(); outfile.close(); //close input and output streams
   return 0;
}
```

Create a makefile and a README file and run your code

Note that **without the README** file the autograder raises an **error**. Note that the README file name should **NOT have any file extension** (e.g., txt or md) otherwise the autograder doesn't accept it. Below you can find a **sample makefile** and a **sample README** file:

Makefile

```
CXX = g++
CXXFLAGS = -Wall -std=c++11 -g

OBJECTS = main.o triangle.o
TARGET = main

$(TARGET): $(OBJECTS)
	$(CXX) $(CXXFLAGS) -o $@ $^

main.o: main.cpp triangle.h
	$(CXX) $(CXXFLAGS) -c main.cpp
```

```
triangle.o: triangle.cpp triangle.h
	$(CXX) $(CXXFLAGS) -c triangle.cpp

clean:
	rm -f $(OBJECTS) $(TARGET)
```

README

```
README for Triangle-Classifier

Niloofar Montazeri, Sept 21, 2025
-----------------------------------------------------
Code files: Makefile, main.cpp, triangle.cpp, triangle.h
Test files: test-input1.txt, test-input2.txt, ideal-output1.txt, ideal-output2.txt
Extra files: README (this file)
-----------------------------------------------------
This is a simple assignment, to understand the overall pipeline we will follow
in our assignments.
-----------------------------------------------------
USAGE:
1) Run "make", to get executable "main".
2) Run "./main <INPUT FILE> <OUTPUT FILE>"

Each line of INPUT FILE should have three numbers for three edges of a triangle.
Each corresponding line of OUTPUT FILE has the class of the corresponding
triangle. The classes are:
"invalid", "equilateral", "isosceles", "scalene"
```

You may find the above files useful, to handle **future I/O** and for **compiling**. Feel free to **copy** and **modify** them for **future assignments.**

Copy/pasting into **README** and **Makefile** as well as copying I/O commands to **main.cpp doesn't raise a flag.**

Fire a terminal to make and run your code

Go the the "**Tools**" menu item, and click on "**Terminal**". This gives you access to your own personal Linux box, configured specifically for this assignment[2].

Change directory to Triangle-Classifier and run **make**:

```
make
```

[2] It is best to do as much as possible through the command line. Even if you're not familiar with Unix, I highly recommend sticking to the command line. You will get better at it, and it will help you greatly in the long run.

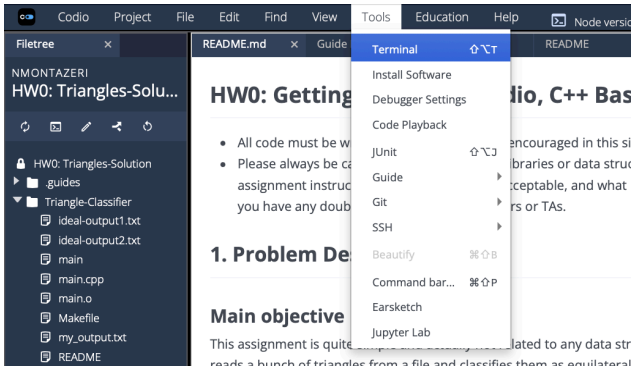
It should create the **executable "main".**

If make has been successful[3], you can now run your **executable (main) with two command line arguments**: the **first is an input file**, the **second is the output file**. For example:

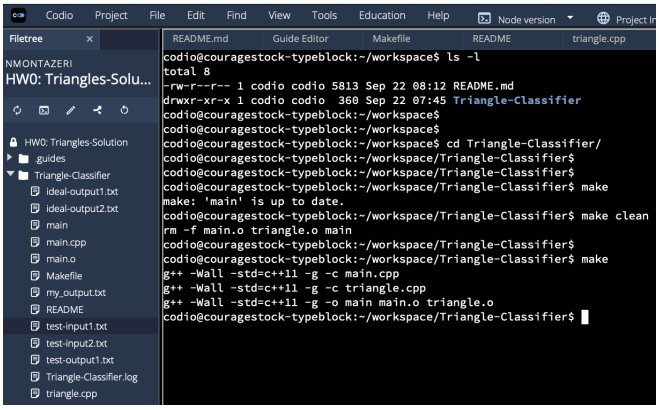
```
./main  test-input1.txt  test-output1.txt
```

Where `test-input1.txt` will have inputs to the **main** program and the file `test-output1.txt` will have the output of it.

We will always provide you with a sample input file as well as the corresponding "ideal" output file so you can do some basic debugging. But there is another way for the initial check too. The checker (next section)!



Accessing Terminal



Terminal Window

Run the checker

Once you are done creating all the files , coding, creating make file and README, running make, running your program with the provided input/ideal output files, you can run ./checkcode.sh (from within the Triangle-Classifier folder). It will check that

- all required files are there
- all executables are created
- your code passes the simple test cases that were originally provided (don't delete them).

Once the starter code runs, your next task is to make sure your code addresses edge cases (next section).

[3] To clean the previous executables before running make again, you can run: *make clean*

Think about Special Cases & Fix the Code

Can you think about some inputs that this code **does NOT** handle correctly? Think about edge cases, exceptions, invalid inputs, etc. List the cases and **sample** sides (a,b,c) values for each case:

- Case 1:
- Case 2:
- …

Now fix the code **(in triangle.cpp)** so it can handle all the cases you listed above.

Note that this step is **VERY important. Ignoring it can have serious consequences:**

- In your tests/homeworks, it will result in **failing on some test cases** and **losing points**.
- In your future **technical interviews**, failing to think about edge cases and addressing them in the code raises a **red flag** for the interviewer!

Run the autograder (sparingly) and fix your code if necessary

If you are pretty sure that you have covered all possible input cases, you can proceed to running the autograder (you can find it on the bottom of the **Guides page**). The autograder will run your code against all our hidden test cases and gives you your current score based o how many test cases you passed. If it doesn't pass all our test cases, you need to think more about other edge cases. Keep in mind that YOU ONLY HAVE **4 ATTEMPTS**, so use the autograder **VERY sparingly**.