Programming assignment for job position at MSI

Dmitriy L. Markovich @ dmmrkovich@gmail.com

April 3, 2017

Contents

1 Triangle challenge

Write a program that will determine the type of a triangle. It should take the lengths of the triangle's three sides as input, and return whether the triangle is equilateral, isosceles or scalene.

The solution shall be stored on github so a link to the solution can be provided for evaluation.

Care should be given to:

- syntax, format (indentation) and comments
- testability of the functions / classes
- potential reusability of part of your code within other domains

Programming language should be C or C++.

A README file shall include a description of the design, its motivation, and instructions for compiling and using it from a linux terminal. The code should compile cleanly (no errors, no warnings) with gcc series 4,5 and 6 (or at least 4.4 and 4.8).

2 Background

Given are the three numbers, that are the lengths of the sides of the triangle — a, b, and c. To form a triangle, the lengths have to satisfy the triangle inequality (each side is shorter than the sum of the other two):

$$a + b > c, \qquad a + c > b, \qquad b + c > a \tag{1}$$

With Eq. 1 satisfied, the lengths define the type that a triangle can belong to. The valid types are:

- **equilateral** — a triangle in which all three sides are equal, $a = b = c$.
- **isosceles** — a triangle that has **two and only two** sides of equal length, $a = b$ or $a = c$ or $b = c$. Two and only two consition is necessary to avoid the overlap with the equilateral type.
- **scalene** — a triangle that has all its sides of different lengths, $a \neq b \neq c$.

3 Project structure

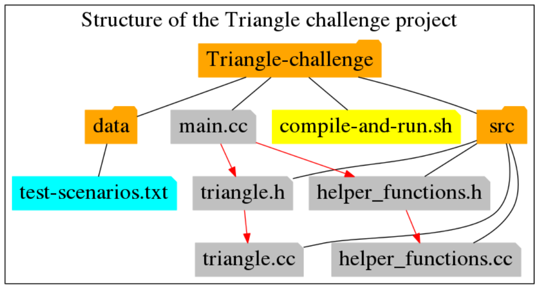

Figure 1. The structure of the project "Triangle challenge". The project consists of the main source file `main.cc` and two header files `triangle.h` (interface of class `Triangle`, see Sec. 5) and `helper_functions.h` (interface of helper functions to load and process triangle data files, see Sec. 6 for details) and their `.cc` source files in the `/src` folder. A default file of test scenarios `test-scenarios.txt` is located in the `/data` folder. The simplest way to add test scenarios is to simply add lines to this file. The script `compile-and-run.sh` compiles and runs the program, reading data from the default triangle data file `/data/test-scenarios.txt`. It may take a command-line argument to specify a non-default name of the triangle data file in `/data`. Upon successful compilation the script will execute the binary with the given triangle data file name and remove the binary afterwards.

4 Compilation and usage

- Make sure `g++` is installed on Your system. If not, try `$ sudo apt-get install g++`.
- Make sure `compile-and-run.sh` is executable. If not, make it executable by running `$ chmod +x compile-and-run.sh`.
- Run the script `$ ./compile-and-run.sh`. It will use the default file `/data/test-scenarios.txt` as the source of different test cases of triangle side lengths input. File will be displayed line-by-line together with the extracted triangle and its type determined by the program. After the execution, the binary will be removed.
- Alternatively, create Your own triangle data file in the `data` folder (for example `/data/new-test.txt`), and run the script as `$ ./compile-and-run.sh new-test.txt` to process it instead of the default one.

5 Class `Triangle`

From the programming perspective it is straightforward to look at a triangle as an object of a certain class. First, the class would have three data members to hold the side lengths of the triangle. The valid lengths would be restricted to be non-zero, positive, non-`nan` and non-`inf` values of type `double`. An additional data member of type `bool` would indicate if all lengths are valid. Since not all combinations of valid lengths can form a triangle , an additional data member of type `bool` would indicate if the lengths satisfy the triangle inequality (Eq. 1). The declaration of such a class is partly presented in List. 1.

Listing 1. A class for a triangle.

```
class Triangle {
 private:
  // the lengths of three sides of the triangle
  double a, b, c;
  // boolean variables showing 1) whether side lengths of the triangle are valid
  // (positive, non-zero, non-nan, non-inf) 2) whether the given lengths form a
  // triangle that satisfies the triangle inequality
  bool side_lengths_are_valid, is_triangle;
 ...
};
```

When an instance of the class `Triangle` contains valid lengths that can form a triangle, the problem breaks down to determining the type of the triangle according to the abovementioned definitions. This task is solved by the constant member function `Triangle::determine_type()`, that is shown in List. 2. The function will first check whether the side lengths are valid and whether they can form a triangle, and return a `"N/A (...)"` string with the accompanying explanation if that is not any of the cases. Next, it will check whether any of the two sides are equal. In case of success, it will check whether all three sides are equal and return `"equilateral"` if succeeded, and `"isosceles"` otherwise. If no pair of sides was equal, the triangle would be classified as `"scalene"`.

Listing 2. A constant member function of class Triangle that determines the type of the triangle.

```
// determines the type of the triangle
std::string Triangle::determine_type() const {
  if (side_lengths_are_valid) {
    if (is_triangle) {
      if (a == b || a == c || b == c) {
        if (a == b && a == c && b == c) {
          // equilateral triangle, all sides are equal
          return "equilateral";
        } else {
          // isosceles triangle, two sides are equal
          return "isosceles";
        }  // End of if (a == b && a == c && b == c) else
      } else {
        // scalene triangle, all sides of different lengths
        return "scalene";
      }  // End of if (a == b || a == c || b == c) else
    } else {
      // valid sides lengths but they can not form a triangle
      return "N/A (lengths can not form a triangle)";
    }  // End of if (is_triangle) else
  } else {
    // sides lengths are not valid
    return "N/A (lengths are not valid)";
  }  // End of if (side_lengths_are_valid) else
}  // End of std::string Triangle::determine_type()
```

6 Testability

A possible scenario to test the performance of `Triangle::determine_type()` would be to create a triangle data file of predefined cases and compare the output of the program to the expected result. The format of the file is a standard csv, where each line (record) contains a single test triangle in the form of three values (fields) separated by commas:

```
6.34,6.34,6.34
6.32,6.34,6.34
6.34,6.31,6.32
...
```

The program would then read the file line-by-line, extract a triangle object from each of them, and classify the triangles. The output of the program to the terminal would be

```
Line: 6.34,6.34,6.34 | Extr trgl: 6.34 6.34 6.34 | Result: equilateral
Line: 6.32,6.34,6.34 | Extr trgl: 6.32 6.34 6.34 | Result: isosceles
Line: 6.34,6.31,6.32 | Extr trgl: 6.34 6.31 6.32 | Result: scalene
...
```

6.1 Log of default test cases

Below is the log of default test cases defined in the default file `/data/test-scenarios.txt`:

```
$ ./compile-and-run.sh
>>>> Running main.cc.bin with default data file for test scenarios.
Line: 6.34,6.34,6.34 | Extr trgl: 6.34 6.34 6.34 | Result: equilateral
Line: 6.32,6.34,6.34 | Extr trgl: 6.32 6.34 6.34 | Result: isosceles
Line: 6.34,6.31,6.32 | Extr trgl: 6.34 6.31 6.32 | Result: scalene
Line: 16.34,6.34,6.34 | Extr trgl: 16.34 6.34 6.34 | Result: N/A (lengths can not form a triangle)
Line: 6.34,6.34,-6.34 | Extr trgl: 6.34 6.34 -6.34 | Result: N/A (lengths are not valid)
Line: 0,3,2 | Extr trgl: 0. 3. 2. | Result: N/A (lengths are not valid)
Line: 1e1,1e1,1e1 | Extr trgl: 10. 10. 10. | Result: equilateral
Line: 1e1,1e1,1e1000 | Extr trgl:  | Result: N/A (failed to split input)
Line: 1e-2,3e-2000,2.5e-1 | Extr trgl: 0.01 0. 0.25 | Result: N/A (lengths are not valid)
Line: 1e-2,-3e-3000,2.5e-1 | Extr trgl: 0.01 -0. 0.25 | Result: N/A (lengths are not valid)
Line: O,1,2 | Extr trgl:  | Result: N/A (failed to split input)
Line: 1,O,2 | Extr trgl:  | Result: N/A (failed to split input)
Line: 1.1,2,O | Extr trgl:  | Result: N/A (failed to split input)
Line: 1,2 | Extr trgl:  | Result: N/A (failed to split input)
Line: 1 | Extr trgl:  | Result: N/A (failed to split input)
Line: A,14,15,13 | Extr trgl:  | Result: N/A (failed to split input)
Line: fdsdf,we,13.4a | Extr trgl:  | Result: N/A (failed to split input)
Line: 14.2a,14.1,15a | Extr trgl:  | Result: N/A (failed to split input)
Line: 14.2,14.1a,15 | Extr trgl:  | Result: N/A (failed to split input)
Line: 1 1 1 | Extr trgl:  | Result: N/A (failed to split input)
Line: 111 | Extr trgl:  | Result: N/A (failed to split input)
Line: 2;-2;2 | Extr trgl:  | Result: N/A (failed to split input)
```