PA 0:
Environment Setup, Debugging, and C++ Refresh

Introduction

The objective of this programming assignment is to help you get ready for the mainstream programming assignments (PA1-5) by covering some introductory refresher topics. Through this assignment, you will complete setting up your own development environment where you will work until the end of the semester, learn the use of debuggers to debug your programs, and recap the core concepts of C/C++ programming.

You are given a simple program, *buggy.cpp*, that computes the area of shapes that has several compile/syntax errors and several bugs that you are going to discover, debug, and fix with the help of three different debugging methods. Once you have corrected the program and committed to the repository, we will use the auto-grading feature of GitHub Classroom to grade your project.

Development Environment Setup

Depending on your host system operating system, there are a few options for the development environment:

1. **Windows 10/11 Only:** WSL2 is the recommended development environment. The tutorial guide can be found here.
2. **Windows 10/11 or x86/AMD64 Mac:** VIrtualBox is the recommended development environment. The tutorial guide can be found here.
3. **M1 Mac Only:** VMWare Fusion is the recommended development environment. The tutorial guide can be found here.
    a. Virtualization on M1 Mac is still relatively new, so you may run into problems. In the event that you are unable to get VMWare Fusion up and running, contact a TA for other options.

You are also welcome to have a clean boot of Linux through dual boot, USB boot, or on a separate machine (remote or local). Using the department Linux servers is strongly advised against.

Once you have your environment set up, you can install relevant packages with **`sudo apt install`** and **`sudo apt-get install`**. The following packages need to be installed:

- g++, gcc, make, gdb, libasan5

The recommended code environment is VSCode (as it has support for GitHub Classroom). Once installed, the recommended extensions are:

- GitHub Classroom, C/C++ Extension Pack, Remote Development (for WSL2)

Debugging Methods

There are three debugging methods presented in this PA. It is recommended you run through PA0 for each method so you can determine which one works best for moving forward with other PAs.

GDB. The first method presented is GDB. In order to use this method, you must include **`-g`** in your compile statement (e.g., **`g++ -g -o buggy buggy.cpp`**). Once compiled, you can run your program in gdb with **`gdb buggy`**. Once GDB is running, to run the executable, simply use the **`run`** command. When/If the program stops due to some error, you can run **`backtrace`** to see where the program stopped and the error it encountered. You can then use the **`break`** command to set a breakpoint before the line the error occurs, and then use the **`print`** command to check values at the breakpoint. You can use **`continue`**/**`step`** to move past the break. GDB Documentation can be found through sourceware.org.

AddressSanitizer. Another useful debugging tool is AddressSanitizer (aka ASan). In order to use ASan, you must include **`-fsanitize=address,undefined`** in your compile statement (e.g. **`g++ -fsanitize=address,undefined -o buggy buggy.cpp`**). You can then run the executable as normal and the tool will print pertinent information about memory issues if an error occurs. The ASan tool is break-at-first-error, meaning multiple runs of AddressSanitizer may be necessary to catch all errors. Tutorials to ASan can be found here, here, and here (here is a longer video) - focus on how ASan is used.

IDE Debugging. The final debugging method is to use the in-built IDE debugging tool. It is required that the GDB package be installed if you are using the VSCode debugger. Debugging reference for VSCode can be found here.

The Buggy Program

The buggy program starter code consists of the following:

- A struct **Point,** which defines a point on the coordinate plane by its x and y value.
- A class **Shape**, which defines a shape formed by a number of different points or vertices (created by the struct Point) on the coordinate plane.
- The function **main()**, which creates two shapes (a triangle and a quadrilateral) using the definitions above.

You are expected to complete the following tasks for the buggy program to be fully functional:

1. Shape's **addPoints(...)** function is expected to take in a single parameter – an unsized Point array called pts.

2. In Shape's **area()** function, the computation of variables **lhs** and **rhs** in their current state is faulty due to incorrect member access of pointers. There are two different methods to fix

such member access; you are expected to use one method to fix **lhs** and the other to fix **rhs**.

3. In **main(),** you are expected to create three Point structs (corresponding to tri1, tri2, and tri3) in three different ways. These points will help construct the Shape object, **tri**. You are also expected to create four Point structs (corresponding to quad1, quad2, quad3, and quad4) in any way you choose. These points will help construct the Shape object, **quad**. Finally, print out the area of the two shapes (tri and quad) created.

Note that your tasks may not just be limited to ones explicitly defined above. Ensure that all dynamically allocated memory is properly cleaned up.

Rubric

There are three tests that check the elements of the program which you have been asked to fix and grades will be assigned as the score you see on GitHub classrooms (under the Actions tab):

1. Compilation (34 pts)
2. Correct output (33 pts)
3. No leaking memory (33 pts)

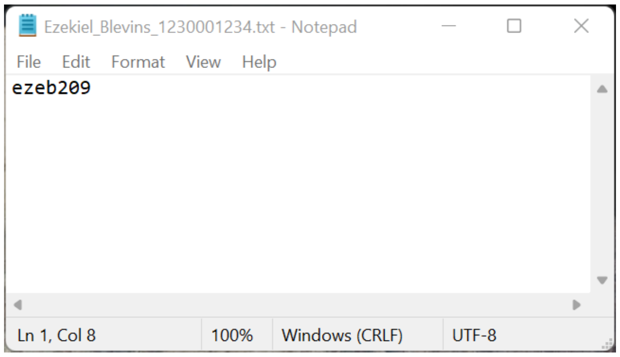
Partial credit will not be assigned for this PA; it is all-or-nothing.

Location of the Code

This assignment will be hosted using the **GitHub classroom** platform. You must visit **https://classroom.github.com/a/w3ctCyiW** to create a repository for your project. Refer to the Lab0B module on canvas for introduction to GitHub Classroom.

What to do when done

Once you have completed the assignment and committed the final code to GitHub Classroom, turn in a text document named *FirstName_LastName_UIN.txt* containing your GitHub user login (example below) and a screenshot of your development environment on Canvas.


Ezekiel_Blevins_1230001234.txt - Notepad
File Edit Format View Help
Ln 1, Col 8
100%
Windows (CRLF)
UTF-8

```
ezeb209
```