CSCI 3081W Workshop 1 - C++ Basics

Spring 2025

Submission

After you complete the code, save it as workshop1.cc and submit it on Canvas under Workshop 1 assignment. If you worked with anyone else, please list their names in the Canvas comments for the submission. Workshop 1 is due Thursday, January 30th at 11:59 pm.

Goal

The goal of the workshop is to make use of the concepts studied in zyBooks as well as problem solving strategies to plan, develop, build, and test a program based on given requirements.

Prerequisites

Be able to SSH into a lab machine via VSCode or your terminal, and make sure your quota is below 1.5GB.
https://github.umn.edu/umn-csci-3081w-s25/FAQ/tree/main/SSH
~or~
Know how to use vole, and make sure your vole quota is below 1.5GB.
(Else, contact csehelp@umn.edu)
https://cse.umn.edu/cseit/self-help-guides/virtual-online-linux-environment-vole

Overview

In this workshop, you will write a C++ program that generates a customizable math quiz. The program will generate random arithmetic problems, accept and validate user input, keep track of the user's score, and provide a summary at the end. This task will help you practice using loops, conditional expressions, random numbers, and basic I/O.

How to Compile and Run

Compile: `g++ workshop1.cc`
Run: `./a.out`
Frequently recompile and test your program as you develop!

Introduction to Waterfall and Requirements

Waterfall Model

The Waterfall Model is a traditional software development methodology that organizes the development process into distinct phases. Each phase has a specific purpose and must be completed before moving to the next. The key phases are:

Requirements: Defining what the software must do. This is the foundation for the entire project.

Design: Planning how the software will meet the requirements. This includes system architecture and detailed designs for each component.

Implementation: Writing the code to build the software according to the design.

Verification: Testing the software to ensure it meets the requirements and works as intended.

Maintenance: Updating and fixing the software after deployment to address new needs or issues.

The most critical aspect of the Waterfall Model is its linear nature. Once a phase is completed, it’s challenging to go back and make changes. Because of this, the Requirements phase is especially important to get right from the start.

Requirements

Requirements are clear, specific, and testable statements describing what the software should do. They guide the entire development process and ensure all team members have a shared understanding of the goals.

Here are the main characteristics of good requirements:

Clear and Unambiguous: Requirements should be straightforward and leave no room for multiple interpretations. Avoid vague terms like "user-friendly" without context.

Testable: You should be able to verify through testing whether the requirement has been met.

Relevant: Requirements should directly relate to the needs of the user or the problem being solved.

Example of a Good Requirement: "The system must accept two integers as input and output their sum."

This requirement is clear because it specifies exactly what the system must do. It is also testable because you can verify its functionality by providing inputs and checking the output.

Why Requirements Matter

They establish a clear goal for the project.

They prevent misunderstandings and miscommunications among developers and stakeholders.

They serve as a benchmark during testing to ensure the software meets expectations.

In today's workshop, you will follow a simplified version of the Waterfall Model. The requirements for your program will be provided, and you will use them to guide your implementation. Ensure you fully understand the requirements before moving on to coding, as this will help you avoid unnecessary rework and confusion later in the process.

Basic Quiz Generator Requirements

1. The program must generate a random math problem (addition, subtraction, multiplication) for the user.
2. The program must prompt the user for their answer.
3. The program must compare the user's answer to the correct answer and provide feedback (correct or incorrect).
4. The program must allow the user to choose the number of questions they want to answer.
5. The program must display the user's total score at the end.

Code Template

```
#include <iostream>
#include <cstdlib>  // For rand() and srand()
#include <ctime>    // For seeding random numbers

int main() {
    // Seed random number generator
    srand(time(0));

    return 0;
}
```

Example Runs from Terminal

Welcome to the Math Quiz Generator!
How many questions would you like to answer? 3
Question 1: 10 * 4 = ? 40
Correct!
Question 2: 8 * 9 = ? 17
Incorrect. The correct answer was 72.
Question 3: 4 - 10 = ? -6
Correct!
You answered 2 out of 3 questions correctly.

Welcome to the Math Quiz Generator!
How many questions would you like to answer? 2
Question 1: 10 * 6 = ? 60
Correct!
Question 2: 1 * 9 = ? 9
Correct!
You answered 2 out of 2 questions correctly.

Testing and Feedback

Test your program with different inputs. Feel free to do peer testing. You can swap programs with a neighbor and you can test each other's programs. Discuss common issues like handling invalid input, floating-point comparisons, or the use of `rand()`.

Extra Credit

If you choose to do the extra credit and want to get credit for it, you must comment "extra credit" in your Canvas submission comments section.

1. Input Validation: Ensure the user enters valid integers.
2. Difficulty levels: Allow the user to select easy, medium, or hard (modifying the difficulty of the randomly generated problems).
3. Scoring System: Include bonus points for speed of correct answers.