**📖 MTH2008** Scientific Computing Logbook

School of Engineering and Physical Sciences, University of Lincoln

Semester A, 2024

*“I confirm that this logbook is entirely my own work and that all references and quotations, from both primary and secondary sources, have been fully identified and properly acknowledged.”* - William Fayers (27378661).

## 📚 Table of Contents

|  |  |
| --- | --- |
| **Section** | **Page** |
| **Disclaimer** | 1 |
| [**Week 1** – 2024-09-30 to 2024-10-07](#week-1-2024-09-30-to-2024-10-07) | 2 |
| 1.1 [Basic Use: Using Cout](#basic-use-using-cout) | 2 |
| 1.2 [Basic Use: Special Characters](#basic-use-special-characters) | 3 |
| 1.3 [Basic Use: Pascals Triangle](#basic-use-pascals-triangle) | 4 |
| *1.4* [*Pascal Triangle: Extended Project*](#pascal-triangle-extended-project) | 5 |
| [**Week 2** – 2024-10-07 to 2024-10-14](#week-2-2024-10-07-to-2024-10-14) | 6 |
| 2.1 [Basic Use: Variables And Data Types](#basic-use-variables-and-data-types) | 6 |
| 2.2 [Basic Use: Input And Math](#basic-use-input-and-math) | 7 |
| *2.3* [*Coursework: Calculator*](#coursework-calculator) | 8 |
| [**Week 3** – 2024-10-14 to 2024-10-21](#week-3-2024-10-14-to-2024-10-21) | 9 |
| 3.1 [Basic Use: Data Size](#basic-use-data-size) | 9 |
| 3.2 [Libraries: Using Cmath](#libraries-using-cmath) | 10 |
| *3.3* [*Coursework: Trig Calculator*](#coursework-trig-calculator) | 11 |
| [**Week 4** – 2024-10-21 to 2024-10-28](#week-4-2024-10-21-to-2024-10-28) | 12 |
| 4.1 [Control Structures: If Else Statements](#control-structures-if-else-statements) | 12 |
| 4.2 [Control Structures: Input Validation](#control-structures-input-validation) | 13 |
| 4.3 [Control Structures: Simple Calculator](#control-structures-simple-calculator) | 14 |
| *4.4* [*Coursework: Quadratic Calculator*](#coursework-quadratic-calculator) | 15 |
| [**Week 5** – 2024-10-28 to 2024-11-04](#week-5-2024-10-28-to-2024-11-04) | 16 |
| 5.1 [Control Structures: Switch Statements](#control-structures-switch-statements) | 16 |
| 5.2 [Control Structures: While Loops](#control-structures-while-loops) | 17 |
| 5.3 [Algorithms: Babylonian Square Root](#algorithms-babylonian-square-root) | 18 |
| *5.4* [*Coursework: Variable Manipulation*](#coursework-variable-manipulation) | 19 |
| [**Week 6** – 2024-11-04 to 2024-11-11](#week-6-2024-11-04-to-2024-11-11) | 20 |
| 6.1 [Control Structures: For Loops](#control-structures-for-loops) | 20 |
| 6.2 [Functions: Introduction](#functions-introduction) | 21 |
| [**Week 7** – 2024-11-11 to 2024-11-18](#week-7-2024-11-11-to-2024-11-18) | 22 |
| 7.1 [Pointers: Introduction](#pointers-introduction) | 22 |
| 7.2 [Pointers: Quadratic Calculator](#pointers-quadratic-calculator) | 23 |
| [**Week 8** – 2024-11-18 to 2024-11-25](#week-8-2024-11-18-to-2024-11-25) | 24 |
| 8.1 [Applications: Quadratic Calculator](#applications-quadratic-calculator) | 24 |
| 8.2 [Applications: Maclaurin](#applications-maclaurin) | 25 |
| 8.3 [Applications: Array Copy Check](#applications-array-copy-check) | 26 |
| [**Week 9** – 2024-11-25 to 2024-12-02](#week-9-2024-11-25-to-2024-12-02) | 27 |
| 9.1 [File Handling: Overview](#file-handling-overview) | 27 |
| [**References**](#references) | 28 |

## **Week 1** – 2024-09-30 to 2024-10-07

**What did you learn in the lab tasks this week?**

In the lab session this week we focused on outputting to a data stream and handling special characters. In order to output to a data stream, I learned we use cout to declare console output and insert data into this with the insertion operator, << - this differs from Python’s print function, but is essentially how it works. This is because C++ is a lower level language, just above assembly code, and is thus more lightweight and efficient, e.g. you don’t need input/output capabilities with all programs, as not all programs need a command-line interface, hence we include the line #include <iostream>to access these stream commands.

Additionally, we explored special characters (\n, \a, \b, \f, \r, and \t), escape sequences (\\, \', \", and \?), and the line terminator (endl). The special characters and escape sequences are similar to those used in Python, but the line terminator was completely new for me. Functionally, it seems to act the same as a newline character, but works by ending the data stream started by cout.

**How well did I learn it?**

I learned everything pretty quickly, making parallels with Python quite often. It’s been really fun learning a completely new, more low-level language and deepening my understanding of programming in general. I did some extra work outside of my lab session, too, which definitely helped me reinforce any new knowledge and understand C++ even quicker.

**How does my solution compare with the official solution?**

The official solution for Task 3 adds an additional explanation of how Pascal’s triangle is obtained, as well as uses a simplified solution using spaces rather than tab characters.

**How can I extend the concepts used in the tasks to form a new project**

In my extra work this week, I’ve extended the concepts used to make Pascal’s triangle by changing the logic to a more loop-based system so that I can use render any number of rows. I also tried an alternative to using the tab characters, setd::setw. Not only this, but I decided to take this number of rows as a user input with std::cin and validated the input, throwing an exception if invalid.

I also researched C++ best practices and styling guidelines, referencing documents like Google Style Guides [[1]](#references) and Doxygen [[2]](#references). The main points are learned from this were: consistent comment style (I opted for /\* and /\*\*), function return rationale (i.e. when to use void), explicitly returning 0 in a main function (to tell the console it ran correctly), docstring placement (before a function, contrasting with Python), and function declaration (placing the opening curly bracket on the same line).

Finally, with the project I already explained, I explored functions, for loops, conditional statements, exception handling, and some basic libraries: cmath, stdexcept, and iomanip.

**Lesson Review Question Answers**

1. Facilitates complex simulations; differs by enabling exploration beyond physical limits.
2. Climate modeling, drug discovery, astrophysics simulations.
3. First computer-assisted proof; raises questions about proof validity.
4. Cost/feasibility issues; safety concerns.
5. From serial to parallel computing; enhances computational power for complex problems.
6. Serial: sequential processing; Parallel: simultaneous processing.
7. Physical limits in miniaturization; heat dissipation, quantum effects.
8. FLOPS: Floating Point Operations Per Second; measures computational performance.
9. Binary: two digits (0, 1); used for electronic circuitry efficiency.
10. Encapsulation, inheritance, polymorphism.

Multiple choice: BCCBCCBBBA

### Basic Use: Using Cout

#### Task 1.5

The first bit of C++ code I’ve ever written!

cout << "Welcome to C++ Programming\n";

### Basic Use: Special Characters

#### Task 2.3

The bell character \a makes a sound in some environments.

cout << "With bell character: Welcome to C++ Programming\a" << "---\n";

The backspace character the last character.

cout << "With backspace character: Welcome to C++ Programming\b" << "---\n";

The return character to the start of the line.

cout << "With return character: Welcome to C++ Programming\r" << "---\n";

The tab character a tab space.

cout << "With tab character: Welcome to C++ Programming\t" << "---\n";

#### Task 2.4

Removing the backslash before the single quote causes a syntax error.

cout << "Escaped characters: Is the symbol \\n called \' Newline \' \? \n";

#### Task 2.5

The line terminator endl starts a new line.

cout << "A rose by any other name would smell as sweet";  
cout << endl;

### Basic Use: Pascals Triangle

/\*\*  
 \* @file l03-basic\_use-pascals\_triangle.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Exploring cout and special characters to print Pascal's Triangle.  
 \* @version 0.1.0  
 \* @date 2024-09-30  
 \*   
 \* @copyright Copyright (c) 2024  
 \*   
 \*/  
#include <iostream>  
  
using namespace std;  
  
  
/\*\*  
 \* @brief Main function to run the program.  
 \*   
 \* @return int Success or failure (0 or 1).  
 \*/  
int main() {  
 /\* Print Pascal's Triangle up to n=5 with cout and the tab character to center the numbers \*/  
 cout << "n\t\t\t\t Pascal's Triangle up to n=5\n";  
 cout << "0\t\t\t\t\t\t1\n";  
 cout << "1\t\t\t\t\t1\t\t1\n";  
 cout << "2\t\t\t\t1\t\t2\t\t1\n";  
 cout << "3\t\t\t1\t\t3\t\t3\t\t1\n";  
 cout << "4\t\t1\t\t4\t\t6\t\t4\t\t1\n";  
 cout << "5\t1\t\t5\t\t10\t\t10\t\t5\t\t1\n";  
  
 cout << "\t\t\t\t Made by William Fayers\n";  
 /\*\*  
 \* ANSWER (Task 3.1):  
 \* I used the tab character, `\t` to center the numbers in the Pascal's Triangle, along with  
 \* `cout` to print the numbers. This ended up with a fairly nice looking Pascal's Triangle.  
 \*/  
  
 /\* Return 0 to indicate the program ran successfully \*/  
 return 0;  
}

### Pascal Triangle: Extended Project

/\*\*  
 \* @file e01-pascal\_triangle-extended\_project.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Outputs Pascal's triangle to the console, extending the concepts from lab 1.  
 \* @version 0.1.0  
 \* @date 2024-10-01  
 \*   
 \* @copyright Copyright (c) 2024  
 \*   
 \*/  
  
#include <iostream> /\* For std::cout, std::cin \*/  
#include <cmath> /\* For std::ceil, std::pow \*/  
#include <stdexcept> /\* For std::invalid\_argument \*/  
#include <iomanip> /\* For std::setw \*/  
  
  
const int MAX\_ROWS = 29;  
const std::string ERROR\_MESSAGE = "Error: rows must be between 0 and "  
 + std::to\_string(MAX\_ROWS) + "!";  
  
/\*\*  
 \* @brief Outputs Pascal's triangle to the console.  
 \*   
 \* @details Calculates Pascal's triangle iteratively using the formula:  
 \* \f[  
 \* C(n, k+1) = C(n, k) \times \frac{n - k}{k + 1}  
 \* \f]  
 \* where \f$C(n, k)\f$ is the coefficient at row \f$n\f$ and column \f$k\f$.  
 \* Ensures the triangle is centered in the console by calculating the maximum  
 \* width of the triangle and the maximum width of the coefficients, using the  
 \* power of 2 to estimate the width of the largest coefficient.  
 \*   
 \* @note Designed to test the use of output streams, loops, error handling, libraries,  
 \* and functions in C++. Extends the concepts from lab 1, exploring new concepts  
 \* like using the `std::setw` function to set the width of the output instead of  
 \* using the tab character.  
 \*   
 \* @param rows The number of rows to output in Pascal's triangle. Must be non-negative  
 \* and less than or equal to MAX\_ROWS.  
 \*  
 \* @throw std::invalid\_argument Throws an error if rows is less than 0 or greater than MAX\_ROWS.  
 \*  
 \* @warning Since the coefficients in Pascal's triangle can grow very large, this  
 \* function may not work as expected for rows greater than 29.  
 \*  
 \* @example  
 \* \code{.cpp}  
 \* output\_pascals\_triangle(4);  
 \* \endcode  
 \*  
 \* Expected Output:  
 \* \code{.plaintext}  
 \* 1  
 \* 1 1  
 \* 1 2 1  
 \* 1 3 3 1  
 \* 1 4 6 4 1  
 \* \endcode  
 \*/  
void output\_pascals\_triangle(int rows) {  
 /\* Throw an error if rows is less than 0 or greater than MAX\_ROWS. \*/  
 if (rows < 0 || rows > MAX\_ROWS) {  
 throw std::invalid\_argument(ERROR\_MESSAGE);  
 }  
  
 /\* Estimate the width of the largest coefficient using the power of 2 \*/  
 int estimated\_max\_coefficient = std::ceil(std::pow(2, rows));  
 int max\_coefficient\_width = std::to\_string(estimated\_max\_coefficient).length() + 1;  
  
 /\* Calculate the maximum width of the triangle. \*/  
 int max\_triangle\_width = max\_coefficient\_width \* (rows + 1);  
  
 /\* Loop through each row in Pascal's triangle. \*/  
 for (int row = 0; row <= rows; row++) {  
 /\* Set the first coefficient to 1 \*/  
 int coefficient = 1;  
  
 /\* Add leading spaces to center the triangle. \*/  
 int leading\_spaces = (max\_triangle\_width - (max\_coefficient\_width \* (row + 1))) / 2;  
 std::cout << std::string(leading\_spaces, ' ');  
  
 /\* Loop through each column in the current row. \*/  
 for (int column = 0; column <= row; column++) {  
 /\* Output the current coefficient. \*/  
 std::cout << std::setw(max\_coefficient\_width) << coefficient;  
  
 /\* Calculate the next coefficient. \*/  
 coefficient = coefficient \* (row - column) / (column + 1);  
 }  
  
 /\* Move to the next row. \*/  
 std::cout << std::endl;  
 }  
}  
  
/\*\*  
 \* @brief Main function to run the program.  
 \*  
 \* @details This function prompts the user for the number of rows to output  
 \* in Pascal's triangle and calls the output\_pascals\_triangle function.  
 \*  
 \* @return Returns 0 to indicate the program has run successfully.  
 \*/  
int main() {  
 /\* Prompt the user for the number of rows to output. \*/  
 std::cout << "Please enter the number of rows you would like to output (0 to "  
 << MAX\_ROWS << "): ";  
 int rows;  
 std::cin >> rows;  
  
 /\* Output Pascal's triangle to the console. \*/  
 try {  
 output\_pascals\_triangle(rows);  
 } catch (const std::invalid\_argument& e) {  
 std::cout << e.what() << std::endl;  
 }  
  
 /\* Output a goodbye message to the user. \*/  
 std::cout << "Made by William Fayers :)" << std::endl;  
  
 /\* Return 0 to indicate the function has run successfully. \*/  
 return 0;  
}

## **Week 2** – 2024-10-07 to 2024-10-14

**What did you learn in the lab tasks this week?**

This week we explored variables and their data types a bit more, as well as how to use inputs and mathematics to prototype a simple calculator.

**How well did I learn it?**

As I already spent last week diving into C++ a lot more, this week was very simple, so I tried to extend the task solutions slightly with error handling etc.

Making great progress so far!

**How does my solution compare with the official solution?**

Task 2.2’s first offical solution differs the most by inputting the two numbers in one line, rather than separately.

The second official solution to Task 2.2 is a little bit more different, simplifying the programming techniques used. For example, it doesn’t handle any errors and doesn’t use a separate functionto handle any calculations.

**How can I extend the concepts used in the tasks to form a new project**

To extend these concepts, I could create a fully functional calculator with verbose error/input handling, which could even be extended to evaluate simple LaTeX expressions.

### Basic Use: Variables And Data Types

#### Task 1.2

Declare and initalise the variables

int total\_children = 10;  
int candies\_per\_child = 5;  
int extra\_candies = 10;

Output the starting values and the total number of candies

std::cout << "HALLOWEEN CANDY CALCULATOR" << std::endl;  
std::cout << "Total Children: " << total\_children << std::endl;  
std::cout << "Candies per Child: " << candies\_per\_child << std::endl;  
std::cout << "Extra Candies: " << extra\_candies << std::endl;  
std::cout << "---" << std::endl;  
std::cout << "Total Candies: " << total\_candies << std::endl;

Return 0 to indicate the program ran successfully

return 0;

### Basic Use: Input And Math

#### Task 2.1

Calculate the quotient on top of the given code for completion.

result = number1 / number2;  
std::cout << "Quotient: " << result << std::endl;

#### Task 2.3

Declare variables to store the masses and distance.

float mass1, mass2, distance;

We then input the masses and distance.

std::cout << "Enter the mass of the first object (kg): ";  
std::cin >> mass1;  
std::cout << "Enter the mass of the second object (kg): ";  
std::cin >> mass2;  
std::cout << "Enter the distance between the objects (m): ";  
std::cin >> distance;

And finally we call the function, catching any exceptions that occur.

try {  
float force = calculate\_gravitational\_force(mass1, mass2, distance);  
std::cout << "Gravitational Force: " << force << " N" << std::endl;  
} catch (std::invalid\_argument& error) {  
std::cerr << "Error: " << error.what() << std::endl;  
}

### Coursework: Calculator

#### Task 1.1

Input the radius of the circle.

float radius;  
input\_number(radius, "Enter the radius of the circle: ");

Calculate the area and circumference of the circle.

float area = M\_PI \* radius \* radius;  
float circumference = 2 \* M\_PI \* radius;  
std::cout << "Area of a circle with radius " << radius << " = " << area << std::endl;  
std::cout << "Circumference of a circle with radius " << radius << " = " << circumference  
<< std::endl;

#### Task 1.2

Input the principal amount, rate of interest, and time period.

float principal, rate, time;  
input\_number(principal, "Enter the principal amount: ");  
input\_number(rate, "Enter the rate of interest (%): ");  
input\_number(time, "Enter the time period (years): ");

Calculate the simple interest.

float simple\_interest = (principal \* rate \* time) / 100;  
std::cout << "Simple Interest on a principal amount of " << principal << " at a rate of "  
<< rate << "% for " << time << " years = " << simple\_interest << std::endl;

#### Task 2.1

False, they are stored as binary numbers.

return 0;  
}

#### Task 2.2

Device.

#### Task 2.3

.cpp extension.

#### Task 2.4

Translate source code into machine readable code (or exectuable program) that the computer can run.

#### Task 2.5

8 bits.

#### Task 2.6

Hexadecimal (base 16).

#### Task 2.7

.

#### Task 2.8

True.

#### Task 2.9

cout is used without the namespace std, and line 5 is missing a semicolon at the end of the line.

#### Task 2.10

Missing the insertion operator on line 8, correct line is cin >> num1 >> num2;

## **Week 3** – 2024-10-14 to 2024-10-21

**What did you learn in the lab tasks this week?**

We learned more about data types, specifically how they’re handled in memory and their sizes. On top of this, we started using the cmath library for more complex calculations.

**How well did I learn it?**

It was quite simple to learn, and interesting to learn more about the lower-level hardware parts, too. It was slightly confusing answering the questions because of a few minor mistakes in the actual questions and example code, but apart from that I learned it all well!

**How does my solution compare with the official solution?**

The official solutions for Task 2.2 are similar to mine, although handled numbers like 1. instead of 1.0 - this seems very confusing to me. It also wrote each expression in one long line, rather than breaking it across multiple with whitespace. The second expression is also incorrectly evaluated as not a real number in the offical solution.

The final part of this week’s lab also added more than the question asked for, but so did I with error handling!

### Basic Use: Data Size

#### Task 1.1

After reaching the max value of the data type, the number goes to zero.

std::cout << "Original number: " << unsigned\_small\_number << std::endl;  
unsigned\_small\_number++;  
std::cout << "Incremented number (1x): " << unsigned\_small\_number << std::endl;  
unsigned\_small\_number++;  
std::cout << "Incremented number (2x): " << unsigned\_small\_number << std::endl;

#### Task 1.3

Similar to overflow, it loops around as an underflow.

signed\_small\_number = -32768;  
std::cout << "Original number: " << signed\_small\_number << std::endl;  
signed\_small\_number--;  
std::cout << "Decremented number (1x): " << signed\_small\_number << std::endl;  
signed\_small\_number--;  
std::cout << "Decremented number (2x): " << signed\_small\_number << std::endl;

### Libraries: Using Cmath

#### Task 2.1

Calculate the square root of 9.0, which resulted in 3.

std::cout << sqrt(9.0) << std::endl;

Calculate the natural logarithm of 1.1, which resulted in 0.0953102.

std::cout << log(1.1) << std::endl;

Calculate 2.5 raised to the power of 3.0, which resulted in 15.625.

std::cout << pow(2.5, 3.0) << std::endl;

Calculate the ceiling of 4.1, which resulted in 5.

std::cout << ceil(4.1) << std::endl;

Calculate (7 + sqrt(7.0)) / 3.0, which resulted in 3.21525.

std::cout << (7 + sqrt(7.0)) / 3.0 << std::endl;

Calculate the hyperbolic tan of 0.342, which resulted in 0.329262

std::cout << tanh(0.342) << std::endl;

Calculate the remainder of 11.0 divided by 4.0, which resulted in 3.

std::cout << fmod(11.0, 4.0) << std::endl;

Calculate the absolute value of -9, which resulted in 9.

std::cout << abs(-9) << std::endl;

Calculate the (floating) absolute value of -4.5, which resulted in 4.5.

std::cout << fabs(-4.5) << std::endl;

Calculate the floor of 8.8, which resulted in 8.

std::cout << floor(8.8) << std::endl;

Calculate the base-10 logarithm of 1.0, which resulted in 0.

std::cout << log10(1.0) << std::endl;

Calculate the cube root of 10.0, which resulted in 2.15443.

std::cout << cbrt(10.0) << std::endl;

#### Task 2.3

Prompt the user for a value of x.

double x;  
std::cout << "Calculation of the hyperbolic cosecant function..." << std::endl;  
std::cout << "Enter a value for x: ";  
std::cin >> x;

Calculate the hyperbolic cosecant of x using the custom csch function.

double y = csch(x);  
if (!std::isnan(y)) {  
std::cout << "Hyperbolic cosecant of " << x << " is: " << y << std::endl;  
} else {  
std::cout << "Hyperbolic cosecant is undefined for x = 0." << std::endl;  
}

### Coursework: Trig Calculator

#### Task 3.0

Start with a verbose number input function to re-use.

void input\_number(double& number, const std::string& input\_message) {  
while (true) {  
std::cout << input\_message;  
std::cin >> number;  
  
if (std::cin.fail()) {  
std::cerr << "Invalid input. Please enter a valid number." << std::endl;  
std::cin.clear();  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
} else {  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
return;  
}  
}  
}

Implement the main program loop with the atanh and acosh functions.

int main() {  
while (true) {  
double choice;  
std::cout << "Select an operation:\n1. Inverse Hyperbolic Tangent\n"  
<< "2. Inverse Hyperbolic Cosine\n0. Exit\n";  
input\_number(choice, "Enter your choice: ");  
  
if (choice == 0) {  
break;  
}  
  
double x;  
switch (static\_cast<int>(choice)) {  
case 1:  
input\_number(x, "Enter a value for x (between -1 and 1): ");  
std::cout << "The inverse hyperbolic tangent of " << x << " is " << atanh(x)  
<< std::endl;  
break;  
case 2:  
input\_number(x, "Enter a value for x (greater than or equal to 1): ");  
std::cout << "The inverse hyperbolic cosine of " << x << " is " << acosh(x)  
<< std::endl;  
break;  
default:  
std::cerr << "Error: Invalid operation number." << std::endl;  
break;  
}  
}  
  
return 0;  
}

#### Task 3.1

Implement the atanh function, with error checking.

double atanh(double x) {  
if (x < -1 || x > 1) {  
std::cerr << "\nError: The input value must be between -1 and 1." << std::endl;  
return NAN;  
}  
return 0.5 \* log((1 + x) / (1 - x));  
}

#### Task 3.2

Implement the acosh function, with error checking.

double acosh(double x) {  
if (x < 1) {  
std::cerr << "\nError: The input value must be greater than or equal to 1." << std::endl;  
return NAN;  
}  
return log(x + sqrt(x \* x - 1));  
}

#### Task 4.1

bba.

#### Task 4.2

TTTT, although note for the third statement, technically the return statement can be implicit, but it’s always required even if it’s implicitly there.

## **Week 4** – 2024-10-21 to 2024-10-28

**What did you learn in the lab tasks this week?**

This lab we covered conditonal statements, input validation, and how we can apply these topics to a simple calculator.

**How well did I learn it?**

It was quite quick to learn everything because of the extra work I’ve been doing, so yet again, I learned it pretty well!

**How does my solution compare with the official solution?**

Generally, both solutions are the same but mine has more error handling and separates things into more functions. For the final task, the official solution also added another function to be calculated, sech(x), whereas I didn’t.

**How can I extend the concepts used in the tasks to form a new project**

I could extend the trigonometric ideas to make a fully functional trig calculator, with various different trigonometric functions to be evaluated.

### Control Structures: If Else Statements

#### Task 1.1

Copy the function to calculate the first piecewise function.

std::tuple<double, double> evaluate\_function1(double x, double &y) {  
if (x >= 2) {  
std::cout << "x is greater than or equal to 2, so \n";  
std::cout << "y = x^3 + sqrt(x) = ";  
y = pow(x, 3) + sqrt(x);  
} else if (x > 0.5 && x < 2) {  
std::cout << "x is between 0.5 and 2, so \n";  
std::cout << "y = 1 - exp(x) = ";  
y = 1 - exp(x);  
} else {  
std::cout << "x is less than or equal to 0.5, so \n";  
std::cout << "y = |x + 5| = ";  
y = fabs(x + 5);  
}  
  
return std::make\_tuple(x, y);  
}

#### Task 1.3

Evaluate the second piecewise function.

std::tuple<double, double> evaluate\_function2(double x, double &y) {  
if (x < -M\_PI/2) {  
std::cout << "x is less than -pi/2, so \n";  
std::cout << "y = -1 - pi/2 = ";  
y = -1 - M\_PI/2;  
} else if (x >= -M\_PI/2 && x <= 0) {  
std::cout << "x is between -pi/2 and 0, so \n";  
std::cout << "y = sin(x) + x = ";  
y = sin(x) + x;  
} else {  
std::cout << "x is greater than 0, so \n";  
std::cout << "y = x = ";  
y = x;  
}  
  
return std::make\_tuple(x, y);  
}

### Control Structures: Input Validation

#### Task 2.0

Initalises the constant variables.

const double banned\_number = 2.0;  
const double small\_number = 10e-12;

Fetch input from the user.

double x;  
std::cout << "Enter a value for x (other than 2.0): ";  
std::cin >> x;

Calculate the function and output the result.

cout << "When x = " << x << ", the function y = 1/(x^2 - 2) is equal to ";  
double y = 1 / (std::pow(x, 2.0) - 2.0);  
cout << y << endl;

### Control Structures: Simple Calculator

#### Task 3.0

Create a function to input a number, avoiding a banned number.

void input\_number(const std::string &input\_message, double &number, const double &banned\_number){  
const double float\_error\_resolution = 10e-12;  
  
while (true){  
std::cout << input\_message;  
std::cin >> number;  
  
if (std::cin.fail()){  
std::cerr << "Invalid input. Please enter a valid number." << std::endl;  
std::cin.clear();  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
} else if (std::fabs(number - banned\_number) < float\_error\_resolution){  
std::cerr << "Error: x cannot be " << banned\_number << std::endl;  
std::cin.clear();  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
} else {  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
return;  
}  
}  
}

Loop selection until the user exits.

while (true) {  
std::cout << "List of functions:" << std::endl;  
std::cout << "0. Exit" << std::endl;  
std::cout << "1. csch(x)" << std::endl;  
  
int choice;  
std::cout << "Enter the number of the function you would like to use: ";  
std::cin >> choice;  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
std::cout << "---" << std::endl;  
  
double x, y;  
switch (choice){  
case 0:  
std::cout << "Exiting..." << std::endl;  
return 0;  
case 1:  
input\_number("Enter x: ", x, 0.0);  
std::cout << "csch(" << x << ") = ";  
y = 1.0 / std::sinh(x);  
break;  
default:  
std::cerr << "Invalid choice. Please enter a valid number." << std::endl;  
y = NAN;  
break;  
}  
  
if (!std::isnan(y)){  
std::cout << y << std::endl;  
}  
}

### Coursework: Quadratic Calculator

#### Task 5.0

Include the necessary libraries.

#include <cmath>  
#include <iostream>  
#include <limits>  
#include <tuple>

Start with a verbose number input function to re-use.

void input\_number(double& number, const std::string& input\_message) {  
while (true) {  
std::cout << input\_message;  
std::cin >> number;  
  
if (std::cin.fail()) {  
std::cerr << "Invalid input. Please enter a valid number." << std::endl;  
std::cin.clear();  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
} else {  
std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
return;  
}  
}  
}

Calculate the quadratic’s solution.

std::tuple<double, double> solve\_quadratic(const double a, const double b, const double c) {  
double discriminant = pow(b, 2) - 4 \* a \* c;  
if (discriminant < 0) {  
return std::make\_tuple(NAN, NAN);  
}  
  
double x1 = (-b + sqrt(discriminant)) / (2 \* a);  
double x2 = (-b - sqrt(discriminant)) / (2 \* a);  
  
return std::make\_tuple(x1, x2);  
}

Implement the main program loop.

int main() {  
while (true) {  
double a, b, c;  
input\_number(a, "Enter the coefficient of x^2: ");  
input\_number(b, "Enter the coefficient of x: ");  
input\_number(c, "Enter the constant term: ");  
  
std::tuple<double, double> x = solve\_quadratic(a, b, c);  
  
std::cout << "The quadratic equation " << a << "x^2 + " << b << "x + " << c << " = 0 has ";  
  
if (std::isnan(std::get<0>(x))) {  
std::cout << "no real solutions." << std::endl;  
} else if (std::get<0>(x) == std::get<1>(x)) {  
std::cout << "one solution: x = " << std::get<0>(x)  
<< std::endl;  
} else {  
std::cout << "two solutions: x = " << std::get<0>(x)  
<< " and x = " << std::get<1>(x) << std::endl;  
}  
  
char response;  
std::cout << "Would you like to solve another quadratic equation? (y/n): ";  
std::cin >> response;  
  
if (response != 'y') {  
break;  
}  
}  
  
return 0;  
}

## **Week 5** – 2024-10-28 to 2024-11-04

**What did you learn in the lab tasks this week?**

This week we extended the statements we know with the switch and while statements, then applied those concepts to making an algorithm to find the Babylonian square root.

**How well did I learn it?**

I think I learned everything quite well, borrowing from my understanding of other programming languages really helped, too.

**How does my solution compare with the official solution?**

The offical solutions aren’t out yet, but I’ll update this when they are - if they are in time for the logbook submission deadline.

**How can I extend the concepts used in the tasks to form a new project**

As similarly suggested in previous weeks, I could extend these concepts to make a fully functional calculator - maybe I will between semesters!

### Control Structures: Switch Statements

#### Task 1.1

Example switch statement.

void example\_switch\_function() {  
unsigned int incrementor = 3;  
unsigned int number;  
  
switch (incrementor) {  
case 1:  
number = 1;  
break;  
case 2:  
number = 1;  
break;  
case 3:  
number = 2;  
break;  
case 4:  
number = 6;  
break;  
case 5:  
number = 24;  
break;  
default:  
number = 0;  
break;  
}  
  
std::cout << "The number is: " << number << std::endl;  
}

Call the example switch function.

// example\_switch\_function(); // uncomment this line to run the example

#### Task 1.2

Implement a switch statement to select a function.

const double small\_number = 10e-12;  
double x, y;  
int selection;  
  
std::cout << "FUNCTION LIST" << std::endl;  
std::cout << "1. csch(x)" << std::endl;  
std::cout << "2. sech(x)" << std::endl;  
std::cout << "3. coth(x)" << std::endl;  
  
std::cout << "Select a function (1-3): ";  
std::cin >> selection;  
if (selection > 0 && selection < 4) {  
std::cout << "Enter a value for x: ";  
std::cin >> x;  
std::cout << "For x = " << x << ", ";  
}  
  
switch (selection) {  
case 1:  
if (std::abs(x) < small\_number) {  
std::cout << "csch(x) is undefined for x = 0." << std::endl;  
return 1;  
}  
y = 1/sinh(x);  
std::cout << "csch(x) ";  
break;  
case 2:  
y = 1/cosh(x);  
std::cout << "sech(x) ";  
break;  
case 3:  
if (std::abs(x) < small\_number) {  
std::cout << "coth(x) is undefined for x = 0." << std::endl;  
return 1;  
}  
y = 1/tanh(x);  
std::cout << "coth(x) ";  
break;  
default:  
std::cout << "Invalid selection." << std::endl;  
return 1;  
}  
  
std::cout << "= " << y << std::endl;

### Control Structures: While Loops

#### Task 2.1

The precision is set to 10 decimal places.

std::setprecision(10);

The mean is calculated by dividing the sum by the number of values.

double mean = sum / 6;

Calculate the variance and hence standard deviation.

double variance = ((sum\_of\_squares / 6) - (mean \* mean)) \* 6 / 5;  
double standard\_deviation = std::sqrt(variance);

The calculated values are then outputted to the console.

std::cout << "The minimum value is: " << min << std::endl;  
std::cout << "The maximum value is: " << max << std::endl;  
std::cout << "The mean value is: " << mean << std::endl;  
std::cout << "The standard deviation is: " << standard\_deviation << std::endl;

#### Task 2.2

The factorial is calculated using a do-while loop.

do {  
factorial\_result \*= current\_number;  
current\_number++;  
} while (current\_number <= input\_number);

The calculated factorial is outputted to the console.

std::cout << "The factorial of " << input\_number << " is: " << factorial\_result << std::endl;

### Algorithms: Babylonian Square Root

#### Task 3.1

The user is prompted to input a number.

double number = 0;  
std::cout << "Enter a number: ";  
std::cin >> number;

The square root is calculated and outputted to the console.

double square\_root = babylonian\_square\_root(number);  
std::cout << "The square root of " << number << " is " << square\_root << std::endl;

### Coursework: Variable Manipulation

#### Task 6.0

Include the necessary libraries.

#include <cmath>  
#include <iostream>

#### Task 6.1

Declare the variables with their respective data types.

int x = 25;  
int y = 18;  
double pay\_rate = 12.50;  
int first\_number = 10;  
int temporary\_number = first\_number;

#### Task 6.2

Swap the values of x and y using a temporary variable.

int temporary\_x = x;  
x = y;  
y = temporary\_x;

#### Task 6.3

Output the values of x and y and evaluate the expression.

std::cout << "The value of x is: " << x << std::endl;  
std::cout << "The value of y is: " << y << std::endl;  
double evaluated\_expression = (x + 12) / (y - 18 + pow(x, 3)); // Unsure if correct expression  
std::cout << "Substituting these values into the expression (x + 12) / (y - 18 + x^3) gives: "  
<< evaluated\_expression << std::endl;

## **Week 6** – 2024-11-04 to 2024-11-11

**What did you learn in the lab tasks this week?**

…

**How well did I learn it?**

…

**How does my solution compare with the official solution?**

…

**How can I extend the concepts used in the tasks to form a new project**

…

### Control Structures: For Loops

/\*\*  
 \* @file l01-control\_structures-for\_loops.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Explore the for loop in C++  
 \* @version 0.1.0  
 \* @date 2024-11-18  
 \*   
 \* @copyright Copyright (c) 2024  
 \*   
 \*/  
#include <iostream>  
  
void even\_odd\_printer(int count\_limit) {  
 for (int count = 0; count < count\_limit; count++) {  
 if (count % 2 == 0) {  
 std::cout << count << " is even" << std::endl;  
 } else {  
 std::cout << count << " is odd" << std::endl;  
 }  
 }  
}  
  
void geometric\_progression(int count\_limit) {  
 for (int count = 0; count < count\_limit; count++) {  
 std::cout << "2^" << count << " = " << (1 << count) << std::endl;  
 }  
}  
  
void sum\_of\_cubes(int count\_limit) {  
 int sum = 0;  
 for (int count = 0; count < count\_limit; count++) {  
 sum += count \* count \* count;  
 }  
 std::cout << "The sum of the cubes of the first " << count\_limit << " numbers is " << sum << std::endl;  
}  
  
void fibonacci(int count\_limit) {  
 int number\_1 = 0;  
 int number\_2 = 1;  
 for (int count = 0; count < count\_limit; count++) {  
 std::cout << number\_1 << std::endl;  
  
 // Swap the numbers to get the next number in the sequence  
 int temp = number\_1;  
 number\_1 = number\_2;  
 number\_2 += temp;  
 }  
}  
  
int main() {  
 // even\_odd\_printer(10);  
 // geometric\_progression(10);  
 // sum\_of\_cubes(10);  
 fibonacci(10);  
 return 0;  
}

### Functions: Introduction

/\*\*  
 \* @file l02-functions-introduction.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Explore functions in C++  
 \* @version 0.1.0  
 \* @date 2024-11-18  
 \*   
 \* @copyright Copyright (c) 2024  
 \*   
 \*/  
#include <iostream>  
#include <cmath>

## **Week 7** – 2024-11-11 to 2024-11-18

Some questions answers in LAB tasks:

Here are the answers to your questions about pointers and functions in C++:

1. **What is a pointer in C++?** A pointer in C++ is a variable that stores the memory address of another variable. Pointers are used for dynamic memory allocation, arrays, and for passing large structures or classes to functions efficiently.
2. **How do you declare a pointer in C++?** To declare a pointer in C++, you use the asterisk (\*) symbol. For example:

* int\* ptr; // Declares a pointer to an integer

1. **Explain the concept of the “null pointer.”** A null pointer is a pointer that does not point to any valid memory location. It is often used to indicate that the pointer is not currently assigned to any object or variable. In C++, you can assign a pointer to null using:

* int\* ptr = nullptr; // C++11 and later

1. **What is the difference between pass-by-value and pass-by-reference in function parameters?**
   * **Pass-by-value** means that a copy of the actual parameter is passed to the function. Changes made to the parameter inside the function do not affect the original argument.
   * **Pass-by-reference** means that a reference (or alias) to the actual parameter is passed to the function. Changes made to the parameter inside the function will affect the original argument.
2. **Define a function in C++ that calculates the factorial of a number using recursion.** Here is an example of a recursive function to calculate the factorial:

* int factorial(int n) {  
   if (n <= 1) {  
   return 1; // Base case  
   }  
   return n \* factorial(n - 1); // Recursive case  
  }

1. **What is the purpose of the const keyword in function declarations?** The const keyword in function declarations is used to indicate that a parameter or member function will not modify the value of the variable or object it refers to. For example, a const parameter prevents the function from changing the argument passed to it, and a const member function cannot modify the object’s member variables.
2. **How do you return multiple values from a function in C++?** You can return multiple values from a function in C++ using several methods:
   * Using std::pair or std::tuple to return multiple values.
   * Using output parameters (pass-by-reference).
   * Using a struct or class to encapsulate the return values.

* Example using std::pair:
* #include <utility>  
    
  std::pair<int, int> getValues() {  
   return std::make\_pair(1, 2);  
  }

1. **Explain the difference between void\* and other pointer types.** A void\* pointer is a special type of pointer that can point to any data type, but it does not have a specific type associated with it. This means you cannot dereference a void\* pointer directly without casting it to another pointer type. Other pointer types (like int\*, char\*, etc.) are strongly typed, meaning they are associated with a specific data type, allowing for direct dereferencing and type-safe operations.

### Pointers: Introduction

/\*\*  
 \* @file l01-pointers-introduction.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Introduction to pointers in C++  
 \* @version 0.1.0  
 \* @date 2024-11-18  
 \*   
 \* @copyright Copyright (c) 2024  
 \*   
 \*/  
#include <iostream>  
  
void pointer\_example1() {  
 // Declare two integers  
 int number\_1 = 8, number\_2 = 16;  
  
 // Declare two pointers to integers  
 int\* p\_number\_1 = &number\_1;  
 int\* p\_number\_2 = &number\_2;  
  
 // Output addresses of the two integers  
 std::cout << "Address of number\_1: " << &number\_1 << std::endl;  
 std::cout << "Address of number\_2: " << &number\_2 << std::endl;  
  
 // Output stored addresses of the two pointers  
 std::cout << "Address stored in p\_number\_1: " << p\_number\_1 << std::endl;  
 std::cout << "Address stored in p\_number\_2: " << p\_number\_2 << std::endl;  
  
 // Output values stored at the addresses stored in the pointers  
 std::cout << "Value stored at address stored in p\_number\_1: " << \*p\_number\_1 << std::endl;  
 std::cout << "Value stored at address stored in p\_number\_2: " << \*p\_number\_2 << std::endl;  
}  
  
void write\_output(int\* value) {  
 std::cout << "Value stored at address: " << \*value << std::endl;  
}  
  
void compute\_triple(int\* value) {  
 \*value \*= 3;  
}  
  
void pointer\_example2() {  
 int number = 5;  
 int\* p\_number = &number;  
   
 write\_output(p\_number);  
 \*p\_number += 15;  
 write\_output(p\_number);  
 compute\_triple(p\_number);  
 write\_output(p\_number);  
}  
  
int main() {  
 // pointer\_example1();  
 pointer\_example2();  
  
 return 0;  
}

### Pointers: Quadratic Calculator

/\*\*  
 \* @file l02-pointers-quadratic\_calculator.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Quadratic calculator using pointers in C++  
 \* @version 0.1.0  
 \* @date 2024-11-18  
 \*   
 \* @copyright Copyright (c) 2024  
 \*   
 \*/  
#include <iostream>  
#include <cmath>  
  
bool solve\_quadratic(float a, float b, float c, float\* x1, float\* x2) {  
 float discriminant = b \* b - 4 \* a \* c;  
 if (discriminant < 0) {  
 \*x1 = -b / (2 \* a); // Real part  
 \*x2 = sqrt(-discriminant) / (2 \* a); // Imaginary part  
 return false; // No real roots  
 }  
  
 \*x1 = (-b + sqrt(discriminant)) / (2 \* a);  
 \*x2 = (-b - sqrt(discriminant)) / (2 \* a);  
 return true; // Real roots  
}  
  
int main() {  
 float a, b, c;  
 std::cout << "Enter a, b, and c: ";  
 std::cin >> a >> b >> c;  
  
 std::cout << "Solving " << a << "x^2 + " << b << "x + " << c << " = 0" << std::endl;  
  
 float x1, x2;  
 if (solve\_quadratic(a, b, c, &x1, &x2)) {  
 if (x1 == x2) {  
 std::cout << "Root of the auxillary equation: " << x1 << std::endl;  
 std::cout << "Solution of the homogenous equation: y = C1 \* e^(" << x1  
 << "x) + C2 \* x \* e^(" << x1 << "x)" << std::endl;  
 } else {  
 std::cout << "Roots of the auxillary equation: " << x1 << ", " << x2 << std::endl;  
 std::cout << "Solution of the homogenous equation: y = C1 \* e^(" << x1  
 << "x) + C2 \* e^(" << x2 << "x)" << std::endl;  
 }  
 } else {  
 std::cout << "Roots of the auxillary equation: " << x1 << " + " << x2 << "i, " << x1  
 << " - " << x2 << "i" << std::endl;  
 std::cout << "Solution of the homogenous equation: y = C1 \* e^(" << x1  
 << "x) \* cos(" << x2 << "x) + C2 \* e^(" << x1 << "x) \* sin(" << x2 << "x)" << std::endl;  
 }  
  
 return 0;  
}

## **Week 8** – 2024-11-18 to 2024-11-25

…

### Applications: Quadratic Calculator

/\*\*  
 \* @file l01-applications-quadratic\_calculator.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Applying C++ to solve quadratic equations  
 \* @version 0.1.0  
 \* @date 2024-12-07  
 \*  
 \* @copyright Copyright (c) 2024  
 \*  
 \*/  
#include <iostream>  
#include <limits>  
#include <cmath>  
#include <complex>  
#include <stdexcept>  
#include <string>  
#include <sstream>  
#include <vector>  
#include <cctype>  
  
/\*\*  
 \* @brief Class representing a quadratic equation.  
 \*  
 \*/  
class QuadraticEquation  
{  
private:  
 long double a;  
 long double b;  
 long double c;  
  
public:  
 // Constructor  
 QuadraticEquation(long double coeff\_a, long double coeff\_b, long double coeff\_c)  
 {  
 if (std::isnan(coeff\_a) || std::isnan(coeff\_b) || std::isnan(coeff\_c))  
 {  
 throw std::invalid\_argument("Coefficients must be valid numbers.");  
 }  
 a = coeff\_a;  
 b = coeff\_b;  
 c = coeff\_c;  
 }  
  
 /\*\*  
 \* @brief Solves the equation, which could be quadratic or linear.  
 \*  
 \* @return std::pair<std::complex<long double>, std::complex<long double>> Pair of roots.  
 \* If linear, returns one root and NaN for the second.  
 \* If infinitely many solutions, both roots are represented as NaN.  
 \*/  
 std::pair<std::complex<long double>, std::complex<long double>> solve() const  
 {  
 if (a != 0.0L)  
 {  
 // Quadratic case  
 long double discriminant = b \* b - 4.0L \* a \* c;  
  
 // Check for potential overflow/underflow  
 if (std::isinf(discriminant))  
 {  
 throw std::overflow\_error("Discriminant is too large, causing overflow.");  
 }  
  
 // Using complex numbers to handle real and complex roots  
 std::complex<long double> sqrt\_discriminant = std::sqrt(std::complex<long double>(discriminant, 0));  
  
 // Calculate two roots  
 std::complex<long double> two\_a = 2.0L \* a;  
 std::complex<long double> root1 = (-b + sqrt\_discriminant) / two\_a;  
 std::complex<long double> root2 = (-b - sqrt\_discriminant) / two\_a;  
  
 return std::make\_pair(root1, root2);  
 }  
 else if (b != 0.0L)  
 {  
 // Linear case: bx + c = 0 => x = -c / b  
 std::complex<long double> root = -c / b;  
 return std::make\_pair(root, std::complex<long double>(NAN, NAN));  
 }  
 else  
 {  
 if (c == 0.0L)  
 {  
 // Infinite solutions: 0 = 0  
 return std::make\_pair(std::complex<long double>(INFINITY, 0.0L),  
 std::complex<long double>(INFINITY, 0.0L));  
 }  
 else  
 {  
 // No solution: c = 0 where c != 0  
 return std::make\_pair(std::complex<long double>(NAN, NAN),  
 std::complex<long double>(NAN, NAN));  
 }  
 }  
 }  
  
 /\*\*  
 \* @brief Formats the equation as a string.  
 \*  
 \* @return std::string Representation of the equation.  
 \*/  
 std::string to\_string() const  
 {  
 std::ostringstream equation;  
 bool first = true;  
  
 // Format a\*x^2  
 if (a != 0.0L)  
 {  
 equation << a << "x^2 ";  
 first = false;  
 }  
  
 // Format b\*x  
 if (b != 0.0L)  
 {  
 if (!first && b > 0)  
 {  
 equation << "+ ";  
 }  
 else if (b < 0)  
 {  
 equation << "- ";  
 }  
 equation << std::abs(b) << "x ";  
 first = false;  
 }  
  
 // Format c  
 if (c != 0.0L || first)  
 { // Display '0' if all coefficients are zero  
 if (!first && c > 0)  
 {  
 equation << "+ ";  
 }  
 else if (c < 0)  
 {  
 equation << "- ";  
 }  
 equation << std::abs(c) << " ";  
 }  
  
 equation << "= 0";  
 return equation.str();  
 }  
};  
  
/\*\*  
 \* @brief Safely reads a number from the user with validation and limited attempts.  
 \*  
 \* @param prompt Prompt to display to the user.  
 \* @param number Reference to store the validated number.  
 \* @param max\_attempts Maximum number of invalid attempts allowed (default is 5).  
 \* @return bool True if a valid number was read, False otherwise.  
 \*/  
bool read\_number(const std::string &prompt, long double &number, int max\_attempts = 5)  
{  
 int attempts = 0;  
 while (attempts < max\_attempts)  
 {  
 std::cout << prompt;  
 std::string input;  
 std::getline(std::cin, input);  
 std::stringstream ss(input);  
 ss >> number;  
  
 // Check if entire input was consumed and was a valid number  
 if (!ss.fail() && ss.eof())  
 {  
 return true;  
 }  
 else  
 {  
 std::cerr << "Invalid input. Please enter a valid number." << std::endl;  
 attempts++;  
 }  
 }  
 std::cerr << "Maximum invalid attempts reached. Exiting input process." << std::endl;  
 return false;  
}  
  
/\*\*  
 \* @brief Formats a complex number as a string in the form "a + bi".  
 \*  
 \* @param num The complex number to format.  
 \* @return std::string The formatted string.  
 \*/  
std::string format\_complex(const std::complex<long double> &num)  
{  
 if (std::isnan(num.real()) && std::isnan(num.imag()))  
 {  
 return "No Solution.";  
 }  
 if (std::isinf(num.real()) && std::isinf(num.imag()))  
 {  
 return "Infinite Solutions.";  
 }  
  
 std::ostringstream oss;  
 oss << num.real();  
  
 if (num.imag() >= 0.0L)  
 {  
 oss << " + " << num.imag() << "i";  
 }  
 else  
 {  
 oss << " - " << std::abs(num.imag()) << "i";  
 }  
  
 return oss.str();  
}  
  
/\*\*  
 \* @brief Main function to run the Quadratic Solver.  
 \*  
 \* @return int  
 \*/  
int main()  
{  
 std::cout << "=====================================" << std::endl;  
 std::cout << " Quadratic Solver " << std::endl;  
 std::cout << "=====================================" << std::endl;  
  
 std::vector<std::string> equation\_log; // To keep track of all solved equations  
 char continue\_calculating = 'y';  
  
 while (std::tolower(continue\_calculating) == 'y')  
 {  
 try  
 {  
 long double a, b, c;  
  
 // Read coefficients with validation  
 if (!read\_number("Enter coefficient a (a can be 0 for linear equations): ", a))  
 {  
 throw std::runtime\_error("Failed to read coefficient 'a'.");  
 }  
 if (!read\_number("Enter coefficient b: ", b))  
 {  
 throw std::runtime\_error("Failed to read coefficient 'b'.");  
 }  
 if (!read\_number("Enter coefficient c: ", c))  
 {  
 throw std::runtime\_error("Failed to read coefficient 'c'.");  
 }  
  
 // Create a QuadraticEquation object  
 QuadraticEquation equation(a, b, c);  
  
 // Display the equation  
 std::cout << "\nSolving the equation: " << equation.to\_string() << std::endl;  
  
 // Solve the equation  
 auto roots = equation.solve();  
  
 // Prepare solution string  
 std::string solution;  
 if (a != 0.0L)  
 {  
 // Quadratic case  
 long double discriminant = b \* b - 4.0L \* a \* c;  
 if (discriminant > 0.0L)  
 {  
 // Two distinct real roots  
 solution = "x1 = " + format\_complex(roots.first) + ", x2 = " + format\_complex(roots.second);  
 }  
 else if (discriminant == 0.0L)  
 {  
 // One repeated real root  
 solution = "x = " + format\_complex(roots.first);  
 }  
 else  
 {  
 // Complex roots  
 solution = "x1 = " + format\_complex(roots.first) + ", x2 = " + format\_complex(roots.second);  
 }  
 }  
 else  
 {  
 // Linear or no/infinite solutions  
 if (b != 0.0L)  
 {  
 solution = "x = " + format\_complex(roots.first);  
 }  
 else  
 {  
 if (c == 0.0L)  
 {  
 solution = "Infinite solutions.";  
 }  
 else  
 {  
 solution = "No solution.";  
 }  
 }  
 }  
  
 // Display the roots  
 std::cout << "Solution: " << solution << "\n"  
 << std::endl;  
  
 // Log the equation and solution  
 equation\_log.push\_back("Equation: " + equation.to\_string() + " | Solution: " + solution);  
 }  
 catch (const std::exception &ex)  
 {  
 std::cerr << "An error occurred: " << ex.what() << "\n"  
 << std::endl;  
 }  
  
 // Continuation prompt with validation  
 while (true)  
 {  
 std::cout << "Would you like to solve another equation? (y/n): ";  
 std::cin >> continue\_calculating;  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Clear input buffer  
  
 continue\_calculating = std::tolower(continue\_calculating);  
 if (continue\_calculating == 'y' || continue\_calculating == 'n')  
 {  
 break;  
 }  
 else  
 {  
 std::cerr << "Invalid response. Please enter 'y' for yes or 'n' for no." << std::endl;  
 }  
 }  
  
 std::cout << std::endl;  
 }  
  
 // Optionally, display the log of all equations solved  
 if (!equation\_log.empty())  
 {  
 std::cout << "=====================================" << std::endl;  
 std::cout << " Solved Equations " << std::endl;  
 std::cout << "=====================================" << std::endl;  
 for (size\_t i = 0; i < equation\_log.size(); ++i)  
 {  
 std::cout << i + 1 << ". " << equation\_log[i] << std::endl;  
 }  
 std::cout << "=====================================" << std::endl;  
 }  
  
 std::cout << "Thank you for using the Quadratic Solver." << std::endl;  
 return 0;  
}

### Applications: Maclaurin

/\*\*  
 \* @file l02-applications-maclaurin.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Calculating the Maclaurin series of functions using C++  
 \* @version 0.1.0  
 \* @date 2024-12-08  
 \*  
 \* @copyright Copyright (c) 2024  
 \*  
 \*/  
#include <iostream>  
#include <cmath>  
#include <iomanip>  
#include <limits>  
#include <sstream>  
#include <string>  
#include <vector>  
  
// Function 1: \frac{1}{\pi + x}  
// Function 2: \ln(3 + x)  
  
/\*\*  
 \* @brief Function to calculate the Maclaurin series of \frac{1}{\pi + x}.  
 \*  
 \* @param x Value of x.  
 \* @param terms\_to\_calculate Number of terms to calculate.  
 \* @return double Maclaurin series value.  
 \*/  
double maclaurin\_series\_1(double x, int terms\_to\_calculate)  
{  
 double sum = 0.0;  
 for (int term = 0; term < terms\_to\_calculate; ++term)  
 {  
 sum += std::pow(-1, term) \* std::pow(x / M\_PI, term) / M\_PI;  
 }  
 return sum;  
}  
  
/\*\*  
 \* @brief Function to calculate the Maclaurin series of \ln(3 + x).  
 \*  
 \* @param x Value of x.  
 \* @param terms\_to\_calculate Number of terms to calculate.  
 \* @return double Maclaurin series value.  
 \*/  
double maclaurin\_series\_2(double x, int terms\_to\_calculate)  
{  
 double sum = std::log(3.0);  
 for (int term = 1; term <= terms\_to\_calculate; ++term)  
 {  
 sum += std::pow(-1, term + 1) \* std::pow(x / 3.0, term) / term;  
 }  
 return sum;  
}  
  
/\*\*  
 \* @brief Function to display the Maclaurin series of a function.  
 \*  
 \* @param x Value of x.  
 \* @param terms\_to\_calculate Number of terms to calculate.  
 \*/  
void display\_maclaurin\_series(double x, int terms\_to\_calculate)  
{  
 std::cout << "Calculating the Maclaurin series of 1/(pi + x) at x = " << x  
 << " with " << terms\_to\_calculate << " terms." << std::endl;  
 std::cout << "Result: " << maclaurin\_series\_1(x, terms\_to\_calculate) << std::endl;  
 std::cout << "Expected: " << 1.0 / (M\_PI + x) << std::endl;  
  
 std::cout << "Calculating the Maclaurin series of ln(3 + x) at x = " << x  
 << " with " << terms\_to\_calculate << " terms." << std::endl;  
 std::cout << "Result: " << maclaurin\_series\_2(x, terms\_to\_calculate) << std::endl;  
 std::cout << "Expected: " << std::log(3 + x) << std::endl;  
}  
  
/\*\*  
 \* @brief Main function to run the Maclaurin series calculator.  
 \*  
 \* @return int  
 \*/  
int main()  
{  
 display\_maclaurin\_series(0.5, 100);  
}

### Applications: Array Copy Check

/\*\*  
 \* @file l03-applications-array\_copy\_check.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Duplicate an array and check if the copy is correct.  
 \* @version 0.1.0  
 \* @date 2024-12-08  
 \*  
 \* @copyright Copyright (c) 2024  
 \*  
 \*/  
#include <iostream>  
#include <vector>  
#include <algorithm>  
#include <numeric>  
#include <iterator>  
  
/\*\*  
 \* @brief Function to duplicate an array using std::copy.  
 \*  
 \* @param arr Array to duplicate.  
 \* @return std::vector<int> Duplicated array.  
 \*/  
std::vector<int> duplicate\_array(const std::vector<int> &arr)  
{  
 std::vector<int> copy(arr.size());  
 std::copy(arr.begin(), arr.end(), copy.begin());  
 return copy;  
}  
  
/\*\*  
 \* @brief Function to check if two arrays are equal.  
 \*  
 \* @param arr1 First array.  
 \* @param arr2 Second array.  
 \* @return bool True if arrays are equal, False otherwise.  
 \*/  
bool arrays\_equal(const std::vector<int> &arr1, const std::vector<int> &arr2)  
{  
 return arr1 == arr2;  
}  
  
/\*\*  
 \* @brief Main function to duplicate an array and check if the copy is correct.  
 \*  
 \* @return int  
 \*/  
int main()  
{  
 std::vector<int> original\_array = {1, 2, 3, 4, 5};  
 std::vector<int> copied\_array = duplicate\_array(original\_array);  
  
 // Display the original and copied arrays  
 std::cout << "Original Array: ";  
 std::copy(original\_array.begin(), original\_array.end(), std::ostream\_iterator<int>(std::cout, " "));  
 std::cout << "\n";  
  
 std::cout << "Copied Array: ";  
 std::copy(copied\_array.begin(), copied\_array.end(), std::ostream\_iterator<int>(std::cout, " "));  
 std::cout << "\n";  
  
 // Check if the arrays are equal  
 if (arrays\_equal(original\_array, copied\_array))  
 {  
 std::cout << "The copied array is correct." << std::endl;  
 }  
 else  
 {  
 std::cout << "The copied array is incorrect." << std::endl;  
 }  
  
 return 0;  
}

## **Week 9** – 2024-11-25 to 2024-12-02

…

### File Handling: Overview

/\*\*  
 \* @file l01-file\_handling-overview.cpp  
 \* @author William Fayers (william@fayers.com)  
 \* @brief Reading/writing to a file in C++  
 \* @version 0.1.0  
 \* @date 2024-12-08  
 \*  
 \* @copyright Copyright (c) 2024  
 \*  
 \*/  
#include <iostream>  
#include <fstream>  
#include <string>  
#include <vector>  
  
/\*\*  
 \* @brief Function to write a vector of strings to a file.  
 \*  
 \* @param filename Name of the file to write to.  
 \* @param data Vector of strings to write to the file.  
 \*/  
void write\_to\_file(const std::string &filename, const std::vector<std::string> &data)  
{  
 std::ofstream file(filename);  
 if (!file.is\_open())  
 {  
 throw std::runtime\_error("Failed to open file for writing.");  
 }  
  
 for (const std::string &line : data)  
 {  
 file << line << std::endl;  
 }  
  
 file.close();  
}  
  
/\*\*  
 \* @brief Reads a file and returns the contents as a vector of strings.  
 \*  
 \* @param filename Name of the file to read.  
 \* @return std::vector<std::string> Contents of the file.  
 \*/  
std::vector<std::string> read\_from\_file(const std::string &filename)  
{  
 std::vector<std::string> data;  
 std::ifstream file(filename);  
 if (!file.is\_open())  
 {  
 throw std::runtime\_error("Failed to open file for reading.");  
 }  
  
 std::string line;  
 while (std::getline(file, line))  
 {  
 data.push\_back(line);  
 }  
  
 file.close();  
 return data;  
}  
  
/\*\*  
 \* @brief Main function to write/read to a file.  
 \*  
 \* @return int  
 \*/  
int main()  
{  
 std::vector<std::string> data = {  
 "Hello, World!",  
 "This is a test file.",  
 "Writing to a file in C++ is easy.",  
 "This is the last line of the file."};  
  
 std::string filename = "output.txt";  
 write\_to\_file(filename, data);  
  
 std::cout << "Data written to file: " << filename << std::endl;  
  
 std::vector<std::string> read\_data = read\_from\_file(filename);  
  
 std::cout << "Data read from file: " << filename << std::endl;  
  
 for (const std::string &line : read\_data)  
 {  
 std::cout << line << std::endl;  
 }  
  
 return 0;  
}

## 📓 References

[1] “A comprehensive guide to C++ coding standards and best practices,” *C++ Style Guide*, 2024. [Online] Available: https://google.github.io/styleguide/cppguide.html (accessed 2024-10-05).

[2] “Official documentation for Doxygen’s documentation blocks and usage,” *Doxygen Manual - Documentation Blocks*, 2024. [Online] Available: https://www.doxygen.nl/manual/docblocks.html (accessed 2024-10-05).