**📖 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 |
| [**References**](#references) | 12 |

## **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?**

There aren’t any official solutions right now, so I’ll write this next week if they get released.

**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 1.1

Removing /n causes the output to not start a new line after the string.

cout << "Without newline: Welcome to C++ Programming" << "---\n";

#### Task 1.3

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 1.3

The formfeed character a ♀ character, representing a page break.

cout << "With formfeed character: Welcome to C++ Programming\f" << "---\n";

#### Task 1.4

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

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

#### Task 2.1

The line terminator endl starts a new line.

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

#### Task 2.2

There’s no change in output if endl is inserted at the end of the string.

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

### Basic Use: Pascals Triangle

#### Task 1.1

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.

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";

### 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 2024-10-14  
 \*   
 \* @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?**

…

**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**

…

**Coursework Question Answers**

1. True.
2. Device.
3. .cpp extension.
4. Translate source code into machine readable code (or exectuable program) that the computer can run.
5. 8 bits.
6. Hexadecimal (base 16).
7. .
8. True.
9. cout is used without the namespace std, and line 5 is missing a semicolon at the end of the line.
10. Missing the insertion operator on line 8, correct line is cin >> num1 >> num2;

### 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;

#### Task 1.2

unsigned short int can be broken down into three parts:- unsigned: This means the variable can only store positive values, which fits the context of candies, as we cannot have a negative number of candies.- short: This means the variable can store values between 0 and 65,535, which is more than enough for the number of candies we are calculating.- int: This means the variable is an integer, which is the type of data we are working with.Hence, unsigned short int is the most appropriate data type for the variable total\_candies.

unsigned short int total\_candies;

Calculate the total number of candies by multipling the average per child by the total number of children, then adding the extra candies.

total\_candies = total\_children \* candies\_per\_child + extra\_candies;

### Basic Use: Input And Math

#### Task 2.2

To calculate the remainder of two floats without using any library functions, we can cast the result of the division to an integer, multiply it by the second number, and subtract it from the first number. This will give us the remainder of the division. If we used the fmod function from the cmath library, we could calculate the remainder directly without the need for casting. I only found this out after writing the code, though.

result = number1 - (static\_cast<int>(number1 / number2) \* number2);  
std::cout << "Remainder: " << result << std::endl;

#### Task 2.3

We can create a function called calculate\_gravitational\_force that takes the masses of two objects and the distance between them as input parameters. The function calculates the gravitational force between the two objects using the formula:

float calculate\_gravitational\_force(float mass1, float mass2, float distance) {  
const float G = 6.67430e-11;  
if (distance == 0) {  
throw std::invalid\_argument("Distance cannot be zero.");  
}  
return (G \* mass1 \* mass2) / (distance \* distance);  
}

#### 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;

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

**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**

…

**Coursework Question Answers**

Multiple choice: bba. True or false: TTTT (note for the third statement, technically the return statement can be implicit, but it’s always required even if it’s implicitly there).

### 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 1.2

After reaching the max value of the data type, the number goes to -32768. I think this is because it wraps around to the minimum value of the data type when it reaches the maximum. This is called an overflow.

short int signed\_small\_number = 32767;  
std::cout << "Original number: " << signed\_small\_number << std::endl;  
signed\_small\_number++;  
std::cout << "Incremented number (1x): " << signed\_small\_number << std::endl;  
signed\_small\_number++;  
std::cout << "Incremented number (2x): " << signed\_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;

#### Task 1.4

The size of the data types follow a pattern, where the size of the data type is doubled when using a long data type (e.g. long int, long long int, long double). The size of each data type logically increases as the data type can store larger numbers.

std::cout << "Size of an unsigned int: " << sizeof(unsigned int) << " bytes" << std::endl;  
std::cout << "Size of an int: " << sizeof(unsigned\_small\_number) << " bytes" << std::endl;  
std::cout << "Size of a short int: " << sizeof(signed\_small\_number) << " bytes" << std::endl;  
std::cout << "Size of a long int: " << sizeof(long int) << " bytes" << std::endl;  
std::cout << "Size of a long long int: " << sizeof(long long int) << " bytes" << std::endl;  
std::cout << "Size of a double: " << sizeof(double) << " bytes" << std::endl;  
std::cout << "Size of a long double: " << sizeof(long double) << " bytes" << std::endl;  
std::cout << "Size of a float: " << sizeof(float) << " bytes" << std::endl;  
std::cout << "Size of a char: " << sizeof(char) << " bytes" << std::endl;

#### Task 1.5

The maximum and minimum values of the data types are defined in the limits.h header file. The maximum value of a signed data type is always one less than the maximum value of the unsigned data type of the same size. The minimum value of a signed data type is always the negative of the maximum value minus one.

std::cout << "Maximum value of a short int: " << SHRT\_MAX << std::endl;  
std::cout << "Minimum value of a short int: " << SHRT\_MIN << std::endl;  
std::cout << "Maximum value of an unsigned short int: " << USHRT\_MAX << std::endl;  
std::cout << "Minimum value of an unsigned short int: " << 0 << std::endl;  
std::cout << "Maximum value of an int: " << INT\_MAX << std::endl;  
std::cout << "Minimum value of an int: " << INT\_MIN << std::endl;  
std::cout << "Maximum value of an unsigned int: " << UINT\_MAX << std::endl;  
std::cout << "Minimum value of an unsigned int: " << 0 << std::endl;  
std::cout << "Maximum value of a long int: " << LONG\_MAX << std::endl;  
std::cout << "Minimum value of a long int: " << LONG\_MIN << std::endl;

### Libraries: Using Cmath

#### Task 2.3

Implement a custom csch function that calculates the hyperbolic cosecant of a number.

double csch(double x) {  
if (x == 0) {  
std::cerr << "Error: Division by zero." << std::endl;  
return NAN;  
}  
  
double answer = 1 / sinh(x);  
return answer;  
}

#### 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 2.2

Perform the complex calculation (resulting in -0.266116):

double calculation\_1 = (2.0 / 15.0) -  
(((1.0 / 3.0) - (sin(0.5))) / (-5.0 / 13.0)) +  
(((1.0 / 3.0) - (tanh(0.3))) / (-(1.0 / 7.0) - (2.0)));  
std::cout << calculation\_1 << std::endl;

Perform the complex calculation (resulting in 6.17943):

double calculation\_2 = pow(  
abs((-9) \* pow((-5), (-2)) - (1)),  
sin(1.0)  
) \* (  
(pow((4.0 / 2.0), (2)) + pow((1.0 / 3.0), (-2)) - cbrt(-5)) /  
sqrt(17.0 / 3.0)  
);  
std::cout << calculation\_2 << 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));  
}

## 📓 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).