**📖 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) | 2 |
| 1.3 [Basic Use: Pascals Triangle](#basic-use-pascals-triangle) | 2 |
| *1.4* [*Pascal Triangle: Extended Project*](#pascal-triangle-extended-project) | 2 |

## **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 and Doxygen. 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

/\*  
\* PROGRAM: l01-basic\_use-using\_cout.cpp  
\* DESCRIPTION: Use of the cout statment  
\* AUTHOR: William Fayers  
\* DATE: 2024-09-30  
\*/  
  
#include <iostream>  
  
using namespace std;  
  
  
int main()  
{  
 /\* Print a welcome message \*/  
 cout << "Welcome to C++ Programming\n";  
 /\* ANSWER (Task 1.5): The first bit of C++ code I've ever written! \*/  
  
 return 0;  
}

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

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

#### 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 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: n Pascal’s Triangle up to n=5 0 1 1 1 1 2 1 2 1 3 1 3 3 1 4 1 4 6 4 1 5 1 5 10 10 5 1 Made by William Fayers n Pascal’s Triangle up to n=5 0 1 1 1 1 2 1 2 1 3 1 3 3 1 4 1 4 6 4 1 5 1 5 10 10 5 1 Made by William Fayers n Pascal’s Triangle up to n=5 0 1 1 1 1 2 1 2 1 3 1 3 3 1 4 1 4 6 4 1 5 1 5 10 10 5 1 Made by William Fayers

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

#### Task e01-pascal.triangle-extended

@brief Outputs Pascal’s triangle to the console. @details This program extends the concepts from lab 1 and explores new concepts such as error handling, loops, libraries, and functions in C++.

#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 \*/

#### Task Outputs.Pascal’s

@details Calculates Pascal’s triangle iteratively using the formula: C(n, k+1) = C(n, k) 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 output\_pascals\_triangle(4); Expected Output: 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1

const int MAX\_ROWS = 29;  
const std::string ERROR\_MESSAGE = "Error: rows must be between 0 and "  
+ std::to\_string(MAX\_ROWS) + "!";

#### Task Main.function

@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.

std::cout << std::endl;  
}  
}