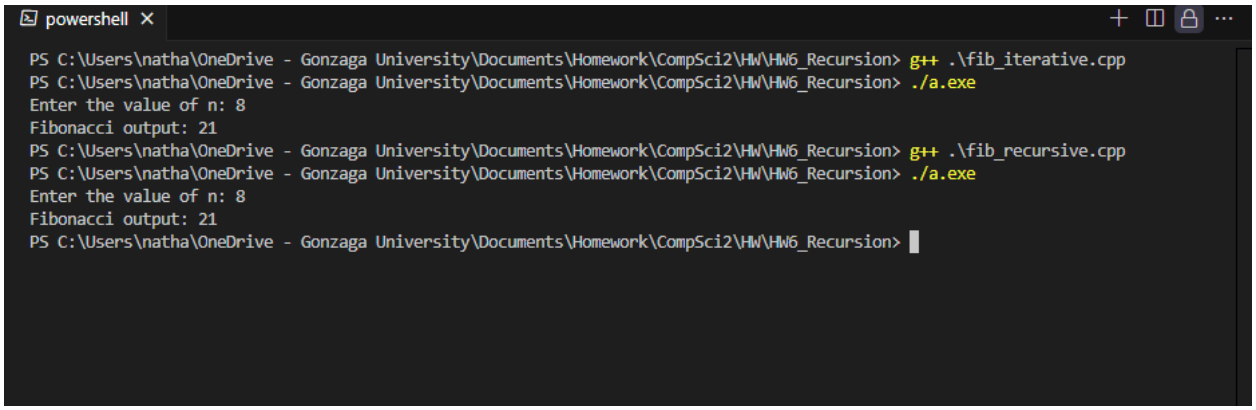
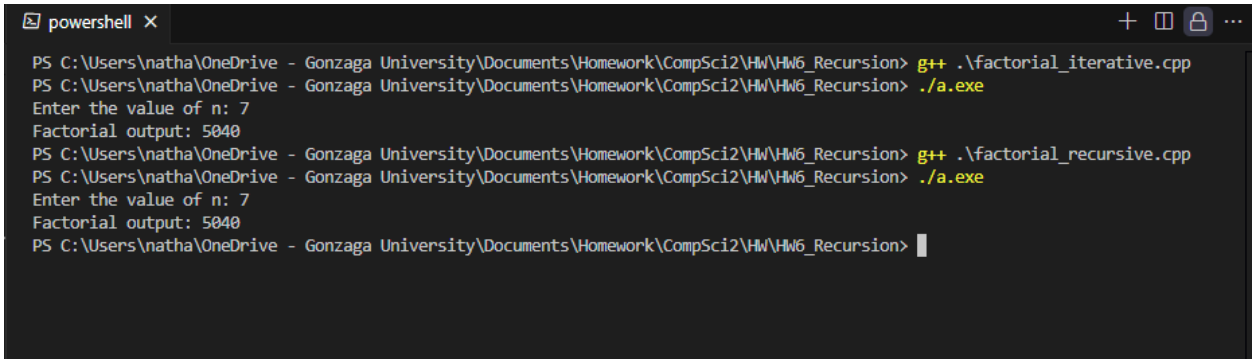
5. Maze solving is a problem that can use recursion effectively. When we look at a standard maze - one entrance and one exit - a common approach to solving mazes using recursion involves the concept of backtracking. This is faster than doing an iterative approach with loops as you wouldn't have to specify an explicit stack to backtrack. To solve a maze using recursion, we start at the maze's entry point and recursively attempt to move in one of the possible directions—up, down, left, or right. If the path leads to a wall or a previously visited cell, you discard that path and return. If the path is open and unexplored, you mark the cell as part of your current route and continue deeper into the maze. When a path fails to yield a solution, the algorithm tries another path. Eventually, this systematic exploration will either discover a path from start to goal or prove that no solution exists. Recursion is well-suited in solving this scenario for several reasons. First, it neatly maps onto the conceptual model of "try a path, and if it doesn't work, backtrack," which aligns perfectly with the definition of recursive calls that attempt smaller subproblems. Additionally, recursion can lead to concise code that's easier to follow conceptually compared to iterative solutions that often require manually managing stacks or queues. However, one downfall of using recursion in solving maze problems is the limitation in technology. For example, to solve a large and complex maze could lead to stack size limitations and cause performance issues.

6. Between the recursive and iterative approaches in implementing the programs, it's difficult to say which is "faster" when comparing each with each other. The reason is, the "speed" at which a computer calculates and compiles the written code to a visual (usually words) we humans will understand is much more complex than just timing the compilation and runtime of the code. Efficiency typically refers to how quickly a program runs or how much memory it uses, but because performance results from a dance between hardware, software, and data patterns, it can be very hard to say definitively why one program is faster than another. Between the recursive and iterative implementations of fibonacci and n! The iterative programming of the Fibonacci calculator was the most complex to read of the 4 programs, as there were many variables to think about going into the right places and getting the logic correct. The easiest to program out of the 4 programs was the recursive factorial program. The iterative programs can cause less problems because there is no stack overflow as iterative programming has explicit values. When we increase the size of n in each program, the big-O complexity of the algorithm dictates how quickly the running time scales. When n is sufficiently large enough, a program can run in exponential time. For recursion, that means each recursive call consumes additional stack space and involves the cost of setting up and tearing down stack frames. When n is large enough, maybe an interactive program would work better. In all, recursion vs iteration and efficiency depend on many factors and would require extensive testing and isolation of parts. It would depend on the problem at hand in deciding whether to use iteration or recursion.


powershell
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> g++ .\fib_iterative.cpp
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> ./a.exe
Enter the value of n: 8
Fibonacci output: 21
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> g++ .\fib_recursive.cpp
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> ./a.exe
Enter the value of n: 8
Fibonacci output: 21
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion>


powershell
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> g++ .\factorial_iterative.cpp
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> ./a.exe
Enter the value of n: 7
Factorial output: 5040
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> g++ .\factorial_recursive.cpp
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion> ./a.exe
Enter the value of n: 7
Factorial output: 5040
PS C:\Users\natha\OneDrive - Gonzaga University\Documents\Homework\CompSci2\HW\HW6_Recursion>