

Peyton Wolf

4/01/2025

Cst-250 Programming in C# ||

Mark Smithers

Assignment 3

### **Loom Video Links**

1. CountToOne: -  
<https://www.loom.com/share/915ab627ee7a407daf970d073c33cafb?sid=2a935009-621a-4d7c-b6cf-3160cc46aac6>
2. Factorial -  
<https://www.loom.com/share/af7350c311224aca959913030cb8f278?sid=9f61c1a1-c1f9-44f9-99c3-a5ecb74f2945>
3. Greatest Common Denominator -  
<https://www.loom.com/share/bcd6c7401bcf4723b6208c0db3a0e3ed?sid=dc35080c-af38-4bdd-9132-4f979fd229d2>
4. Chiasms of the Bible -  
<https://www.loom.com/share/54ad232fee95439abccb7d42ed693e32?sid=90c18159-280a-4445-8c36-4db18d31ccfe>
5. FloodFill-  
<https://www.loom.com/share/95f29969a4c84f3ba932854e2a2daf21?sid=eea628a1-aadb-4a57-95e7-8323518ed387?>??>

GitHub= <https://github.com/KnoxHighStax/CST250/tree/main/Activity3>

### **Programming with Recursion**

```
Please enter an integer. I will do some math and eventually arrive at 1
28
N is 28
N is even. Divide by 2
N is 14
N is even. Divide by 2
N is 7
N is odd. Add 1
N is 8
N is even. Divide by 2
N is 4
N is even. Divide by 2
N is 2
N is even. Divide by 2
N is 1
```

In this demonstration we are utilizing recursion within our algorithm to request the player to input an integer. The user will be asked by the console to input an integer; once the user inputs an integer, if it is odd, it will add 1 to that number to make it even and divisible by 2. From here once the odd number gains 1 it will follow down the same process if the user inputs an even number and will be recursively divided by 2 until an answer of 1 can then be displayed for the user along with the steps of arriving there.

### **Changing Increment**

```

Please enter an integer. I will do some math and eventually arrive at 1
27
Choose how to handle odd numbers
1. Add 1
2. Add 2
3. Subtract 1
3
N is 27
N is odd. Subtract 1
N is 26
N is even. Divide by 2
N is 13
N is odd. Subtract 1
N is 12
N is even. Divide by 2
N is 6
N is even. Divide by 2
N is 3
N is odd. Subtract 1
N is 2
N is even. Divide by 2
N is 1

```

In this screenshot we are demonstrating we have added the ability for the user to have a choice in whether they would like to add to or subtract from the integer that they have previously inputted. We then modified the countToOne function to now take an extra parameter, being “choice”, to help track the user decides to do (if they want to add 1, add 2 or subtract 1).

### ***Division for Even Numbers***

```

Please enter an integer. I will do some math and eventually arrive at 1
381
Choose how to handle odd numbers
1. Add 1
2. Add 2
3. Subtract 1
1
N is 381
N is odd. Add 1
N is 382
N is even. Divide by 3
N is 127
N is odd. Add 1
N is 128
N is even. Divide by 3
N is 42
N is even. Divide by 3
N is 14
N is even. Divide by 3
N is 4
N is even. Divide by 3
N is 1

```

In this screenshot we are just demonstrating a minor adjustment to code them to have its functions to now recursively divide by 3 until we get to 1 for whatever integer the user enters. The only thing we had to do was change the operation from subtract to divide and the number from 2 to 3 and the application is running the same with a few extra steps in the division process; usually depending on what integer the user enters in.

### ***Multiplications for Specific Conditions (Divisible by 5)***

```

Please enter an integer. I will do some math and eventually arrive at 1
9
Choose how to handle odd numbers
1. Add 1
2. Add 2
3. Subtract 1
1
Choose how to handle numbers divisible by 5
1. Multiply by 2
2. Replace with 5
2
N is 9
N is odd. Add 1
N is 10
N is divisible by 5. Replace with 10
N is 5
N is odd. Add 1
N is 6
N is even. Divide by 2
N is 3
N is odd. Add 1
N is 4
N is even. Divide by 2
N is 2
N is even. Divide by 2
N is 1

```

In this screenshot we are demonstrating the ability to add a condition to our code that if at a point a number can be divisible by 5 the user will be given option of specific orders that can be executed. In this instance when divisible by 5 the console will give the user the option to multiply it by 2 or replace the integer as a whole to a predetermined value.

### **Implementing a Counter**

```

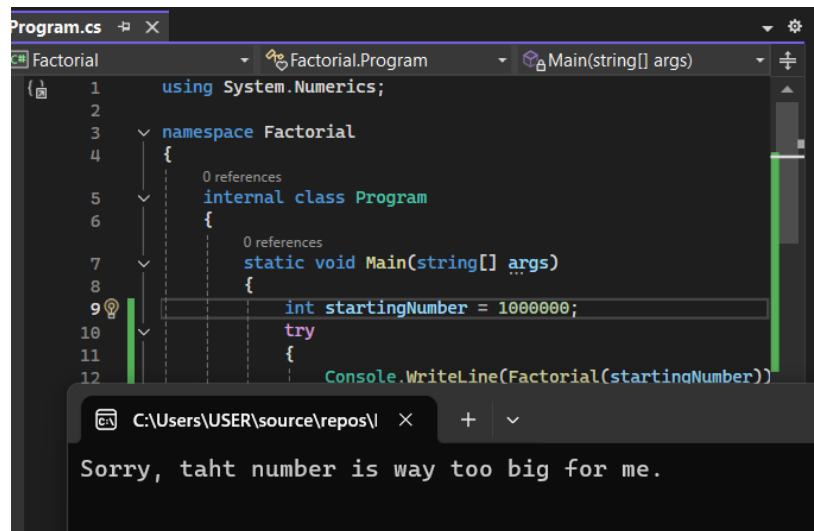
Please enter an integer. I will do some math and eventually arrive at 1
24
Choose how to handle odd numbers
1. Add 1
2. Add 2
3. Subtract 1
1
Choose how to handle numbers divisible by 5
1. Multiply by 2
2. Replace with 5
1
Step1:N is 24
N is even. Divide by 2
Step2:N is 12
N is even. Divide by 2
Step3:N is 6
N is even. Divide by 2
Step4:N is 3
N is odd. Add 1
Step5:N is 4
N is even. Divide by 2
Step6:N is 2
N is even. Divide by 2
Step7:N is 1
Total steps to reach 1: 7

```

In this screenshot we are demonstrating the ability to add a “StepCounter” to our program. Essentially, we initialized the “stepCounter” and then attach it to the return strings to keep

track of each step. Then at the end of the process when we get to 1 the console will display the total number of steps that it took to get to number 1.

### **Factorial Implementing Error Handling**



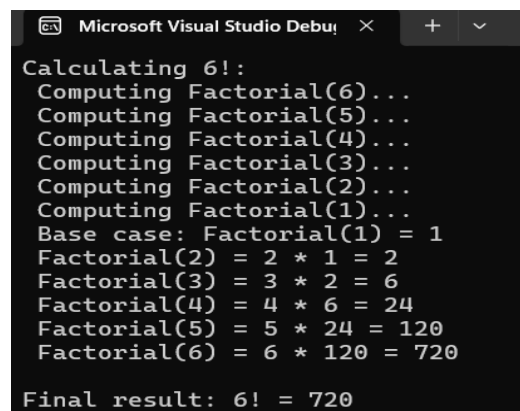
```
Program.cs
1  using System.Numerics;
2
3  namespace Factorial
4  {
5      0 references
6      internal class Program
7      {
8          0 references
9          static void Main(string[] args)
10         {
11             int startingNumber = 1000000;
12             try
13             {
14                 Console.WriteLine(Factorial(startingNumber))
15             }
16             catch { }
17         }
18     }
19 }
```

C:\Users\USER\source\repos\ \ X + v

Sorry, taht number is way too big for me.

In this screenshot we demonstrated the error handling using BigInteger to avoid overflow issues, since factorials grow extremely large and standard types (like int and long) can't handle them. We also added some error handling for any invalid inputs, like negative numbers and extremely large inputs to prevent any crashes or from using excessive computation. Each of which how their own output for the user depending on what they have input.

### **Visual Output of Recursion**



```
Microsoft Visual Studio Debug Console
Calculating 6!:
Computing Factorial(6)...
Computing Factorial(5)...
Computing Factorial(4)...
Computing Factorial(3)...
Computing Factorial(2)...
Computing Factorial(1)...
Base case: Factorial(1) = 1
Factorial(2) = 2 * 1 = 2
Factorial(3) = 3 * 2 = 6
Factorial(4) = 4 * 6 = 24
Factorial(5) = 5 * 24 = 120
Factorial(6) = 6 * 120 = 720
Final result: 6! = 720
```

In this screenshot we have demonstrated the modification to our previous code to allow for visually tracing each recursive step for the user to be able to visualize, while allowing for error handling of larger numbers. This will help the user to be able to easily see the steps of

the factorial process for whatever number that a user may want to input. Currently though the limit enforced by our error handling will cap out at any number over 100001.

### **Comparing Recursive and Iterative Approaches**

```
Computing Factorial(6)...
Computing Factorial(5)...
Computing Factorial(4)...
Computing Factorial(3)...
Computing Factorial(2)...
Computing Factorial(1)...
Base case: Factorial(1) = 1
Factorial(2) = 2 * 1 = 2
Factorial(3) = 3 * 2 = 6
Factorial(4) = 4 * 6 = 24
Factorial(5) = 5 * 24 = 120
Factorial(6) = 6 * 120 = 720
Recursive time for 10000 4885.4575 ms
Iterative time for 10000 runs: 1.5875 ms
Average per call - Recursive: 0.48854575000000006 ms
Average per call - Iterative: 0.0001587499999999998 ms

[Iterative Calculation for 6!]
Final iterative result: 6! = 720
```

In the screenshot we are demonstrating how we have added an Iterative version of the factorial function using a for loop. We will run both version functions for the recursive and iterative function when the program starts up the functions will repeat each of their calculations 'count' number of times and compare for long it took each version to complete ('ComparePerformance' is the method created for this). Then we displayed the total time it took a single calculation (on average) for each factorial version just taking the already previously display time results and dividing it by the 'count'.

### **Adding User Input for GCD**

```
I will ask you for two numbers. I will calculate the greatest common divisor
Please enter the first integer:12
Please enter the second integer:24
Not yet. 12 / 24 has a remainder of 12
Not yet. 24 / 12 has a remainder of 0
The gcd of 12 and 24 is 12
Press any key to exit...
```

In this screenshot we are demonstrating how we have added the ability for the program it accepts user input so that they can input the two integers to calculate the GCD for. We created a method called 'GetValidInteger' that will prompt the user for a valid input, then use 'int.TryParse' to help safely convert the input of the user without throwing exceptions.

### **Compare Recursive and Iterative Solutions**

```
I will ask you for two numbers. I will calculate the greatest common divisor
Please enter the first integer:24
Please enter the second integer:40
Not yet. 24 / 40 has a remainder of 24
Not yet. 40 / 24 has a remainder of 16
Not yet. 24 / 16 has a remainder of 8
Not yet. 16 / 8 has a remainder of 0
[Recursive] The gcd of 24 and 40 is 8
[Iterative] The gcd of 24 and 40 is 8
Press any key to exit...
```

In the screenshot we have extended our GCD program by adding an iterative approach that helps to find the greatest common divisor by listing all divisor of both numbers and then identifying the largest shared divisor. We still have kept the recursive approach so that the user is able to have a side-by-side view of the same data being computed in two different ways.

### **Extend to Multiple Numbers**

```
I will ask you for two (or more) numbers. I will calculate the greatest common divisor
Enter numbers separated by spaces:
24 40 68 88
Not yet. 24 / 40 has a remainder of 24
Not yet. 40 / 24 has a remainder of 16
Not yet. 24 / 16 has a remainder of 8
Not yet. 16 / 8 has a remainder of 0
Not yet. 8 / 68 has a remainder of 8
Not yet. 68 / 8 has a remainder of 4
Not yet. 8 / 4 has a remainder of 0
Not yet. 4 / 88 has a remainder of 4
Not yet. 88 / 4 has a remainder of 0
[Recursive] The gcd of 24,40,68,88 is 4
[Iterative] The gcd of 24,40,68,88 is 4
Press any key to exit...
|
```

In this screenshot we have extended our previous GCD calculator program to be able to handle multiple numbers. Instead of coding for the user to be prompted for each integer in its own separate line, now the program will accept the integers in a singular line with spaces in between the integer, which splits them into an array and then computes the GCD. When the user only inputs two numbers it will function the same in previous iterations; while with 3 or 4 integers, it will chain the GCD calculations using an 'Aggregate()' that just repeatedly applies the GCD function. We adjusted the output slightly to list all input numbers, but the rest of the code all remains the same at the previous iteration.

## **Testing the Performance**

```
I will ask you for two (or more) numbers. I will calculate the greatest common divisor
Enter numbers separated by spaces:
24 40 44
Not yet. 24 / 40 has a remainder of 24
Not yet. 40 / 24 has a remainder of 16
Not yet. 24 / 16 has a remainder of 8
Not yet. 16 / 8 has a remainder of 0
Not yet. 8 / 44 has a remainder of 8
Not yet. 44 / 8 has a remainder of 4
Not yet. 8 / 4 has a remainder of 0
[Recursive] The gcd of 24,40,44 is 4(took13.1983ms)
[Iterative] The gcd of 24,40,44 is 4(took46.5872ms)
Press any key to exit
```

In this screenshot we have shown how we have added an execution time measurement using 'DateTime' to compare the performance of the recursive and iterative methods. For all the numbers that are inputted by the user, we will now capture the time stamps before and after each GCD calculation and display the time results.

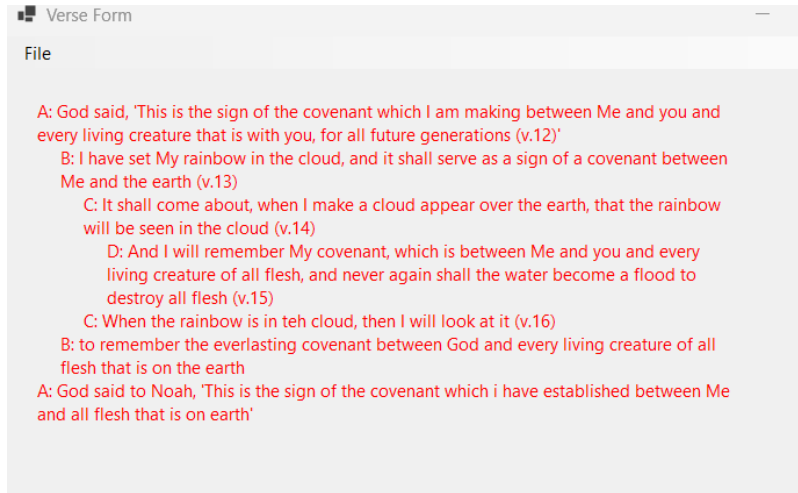
## **Testing Limits and Robustness**

```
Running Testing For Edge Cases:
GCD( -24, -40)
Not yet. 24 / 40 has a remainder of 24
Not yet. 40 / 24 has a remainder of 16
Not yet. 24 / 16 has a remainder of 8
Not yet. 16 / 8 has a remainder of 0
Recursive: 8
Iterative: 8
GCD( -24, 40)
Not yet. 24 / 40 has a remainder of 24
Not yet. 40 / 24 has a remainder of 16
Not yet. 24 / 16 has a remainder of 8
Not yet. 16 / 8 has a remainder of 0
Recursive: 8
Iterative: 8
GCD( 24, -40)
Not yet. 24 / 40 has a remainder of 24
Not yet. 40 / 24 has a remainder of 16
Not yet. 24 / 16 has a remainder of 8
Not yet. 16 / 8 has a remainder of 0
Recursive: 8
Iterative: 8
GCD( 0, 5)
Not yet. 0 / 5 has a remainder of 0
Recursive: 5
Iterative: 5
GCD( 5, 0)
Recursive: 5
Iterative: 5
GCD( 0, 0)
Recursive: 0
Iterative: 0
Testing Complete, you may now continue...
I will ask you for two (or more) numbers. I will calculate the greatest common divisor
Enter numbers separated by spaces:
```

In the screenshot above we have added a comprehensive test suite to help verify the GCD implementation's behavior with an edge case. We created a 'TestGCDCase' helper method that will run both GCD versions on the given inputs and reports the results/errors and a series of test cases in the main that will check for extreme values, zeros and negatives.

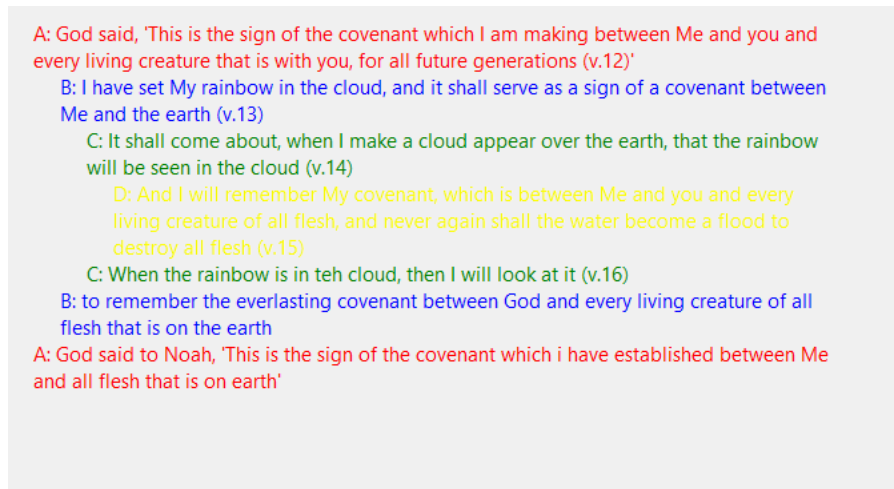


## **Verse Form/ Chiasm**



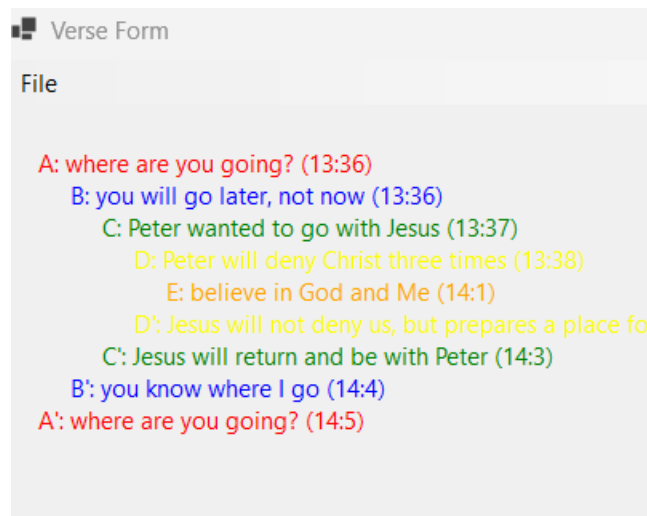
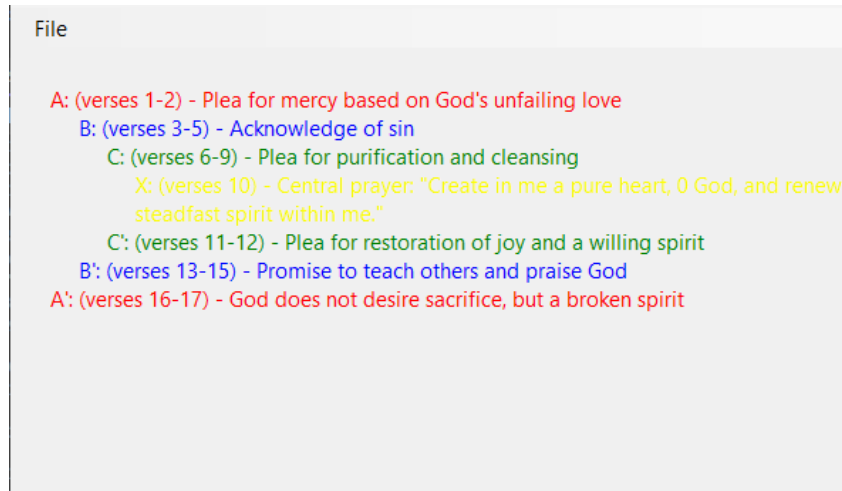
In this screenshot we are demonstrating the first part of our program of a chiasm. We have hard coded in the specific verses we want displayed and then indenting them to make it easier to read. Start with the first verse of the chiasm the DisplayVersesRecursively function will help give each chiasm the proper formatting and indentation and iterate through each verse.

## **Color Coding**



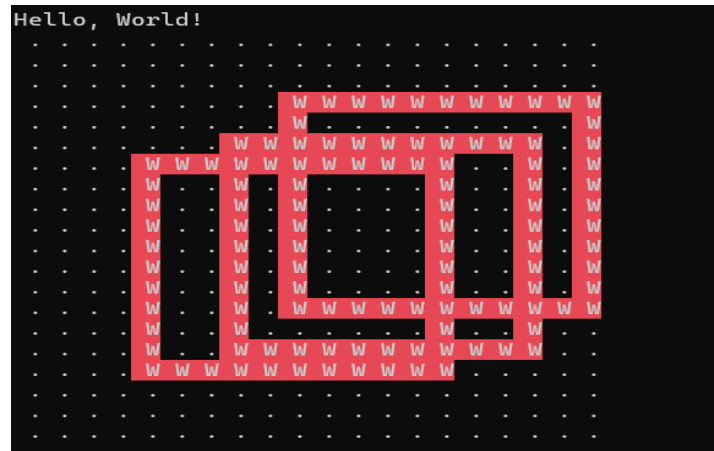
In this screenshot we have just added colors to the different chiasm by using the 'levelColors' array to hold the different distinct colors for each recursion level. With the 'AddVerseToPanel' method the 'level' parameter will help to determine the color index helping to ensure a unique color of each verse.

## **File Open**



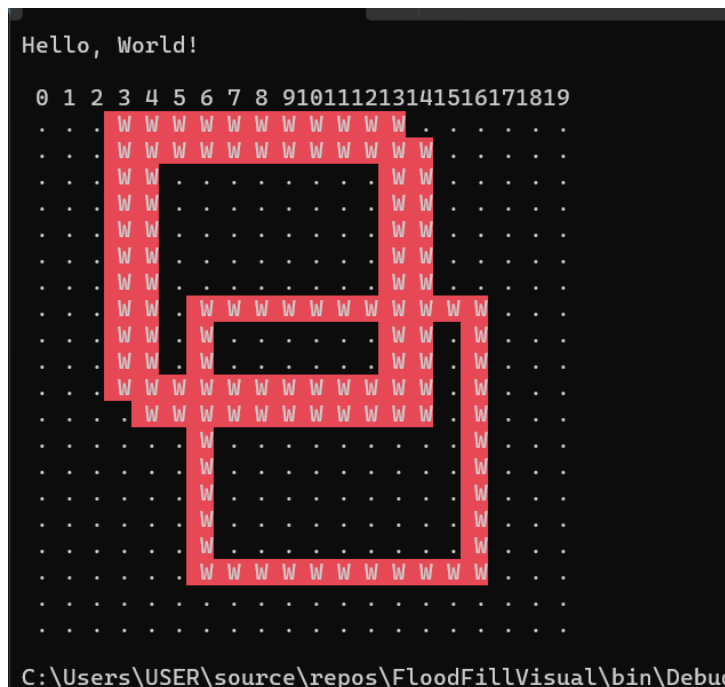
In this screenshot we are showing how we have added a “File > Open” feature to the application, allowing for the user to load verses from a text file instead of relying on hardcoded data. We have included ‘OpenFileDialog’ to select a ‘.txt’ file, a parsing method that also helps to split each line using a pipe ‘|’ delimiter to separate verse labels and the text. Our error handling for this will skip any blank lines and invalid formats.

## **Readability of FloodFill Board**



In this screenshot we are demonstrating for easily readable we made it for the user, turning all the “W” filled with red. For all the “E” we have its place filled with a period “.”, again helping to making it easier to read rather than having a bunch of “E” all over the board, makes it hard to see the “W” even when the color red.

### **Column Numbers**



In this screenshot we just added a for loop to help iterate through each column and assign it a number starting from 1 and continuing for however many columns there are that need to be labeled.

### **Row Numbers**

```

Hello, World!

  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
6  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
7  .  .  W  W  W  W  W  W  W  W  W  W  W  W  .  .  .  .  .
8  .  .  W  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .
9  .  .  W  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .
10 .  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .
11 .  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .
12 .  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .
13 .  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .
14 .  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .
15 .  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .
16 .  .  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  .  .
17 .  .  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  .  .
18 .  .  .  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .
19 .  .  .  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .

```

In this screenshot we just added a for loop to help iterate through each row and assign it a number starting from 1 and continuing for however many columns there are that need to be labeled.

### **Flood Fill**

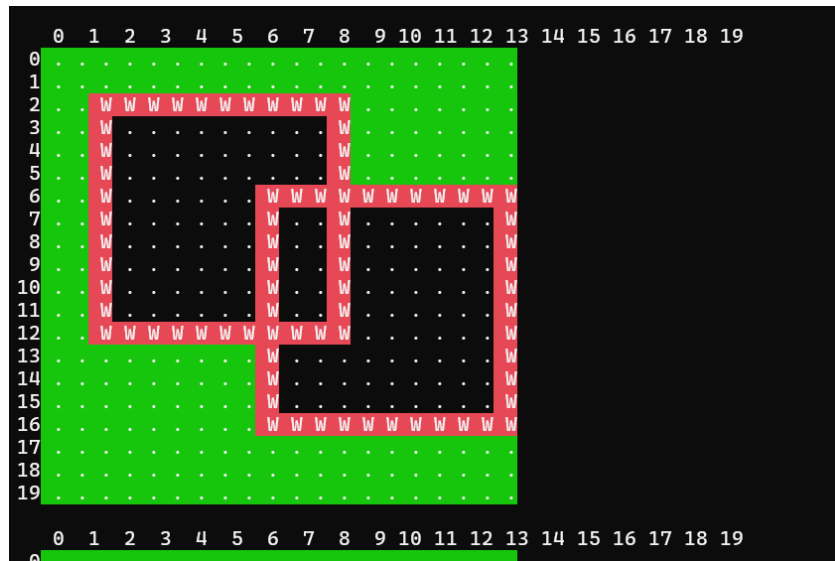
```

North Filling at 17, 0
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
5  .  W  W  W  W  W  W  W  W  W  W  W  W  .  .  .  .  .  .
6  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  .  .  .
7  .  W  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .  .
8  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
9  .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
10 .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
11 .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
12 .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
13 .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
14 .  W  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
15 .  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  .  .  .
16 .  .  .  .  .  .  .  W  .  .  .  .  .  .  .  .  .  W  .  .
17 .  .  .  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .
18 .  .  .  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .
19 .  .  .  .  .  .  .  W  W  W  W  W  W  W  W  W  W  .  .
North Filling at 16, 0 Already filled. Stop.

```

In this screenshot we are demonstrating for we have created and added a “FloodFill” method for the Cells of the Board. The program will check to see if the cell on the board is out of bounds, or to stop at a wall “W”. Then if the cell is not already filled the algorithm will check the north, east, south and then west cell and which everyone if available first is the direction the algorithm will move in to fill the cell.

### **Different Ordering for Recursion**



On this screenshot we are just demonstrating how we have added the ability to adjust the direction of the order at which the algorithm performs the FloodFill. In this iteration we have adjusted the differentOrder to “East, South, North West” and it performs the actions the exact same way as if I went “North, East, South, West”.

### Changing Print Method

```
Key:
E= Empty
W= Wall
F= Filled

What row would you like to start with?
```

In this demonstration we have changed some of the characters for wall (“W”), filled (“F”), and empty (“E”). Updated the user experience like clearing the console display and helping to end to keep the window open. We optimized the color assignments to make them more visually appealing to see.

## Modifying Shapes

```
  0  1  2  3  4  5  6  7  8  9  0  1  2
0  .  .  .  .  .  .  .  .  .  .  .  .  .
1  .  .  .  .  .  .  .  .  .  .  .  .  .
2  .  .  .  .  .  .  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .  .  .  .
5  .  .  .  .  .  .  .  .  .  .  .  .  .
6  .  .  .  .  .  .  .  .  .  .  .  .  .
7  .  .  .  .  .  .  .  .  .  .  .  .  .
8  .  .  .  .  .  .  .  .  .  .  .  .  .
9  .  .  .  .  .  .  .  .  .  .  .  .  .
10 .  .  .  .  .  .  .  .  .  .  .  .  .
11 .  .  .  .  .  .  .  .  .  .  .  .  .
12 .  .  .  .  .  .  .  .  .  .  .  .  .
13 .  .  .  .  .  .  .  .  .  .  .  .  .
14 .  .  .  .  .  .  .  .  .  .  .  .  .
15 .  .  .  .  .  .  .  .  .  .  .  .  .
16 .  .  .  .  .  .  .  .  .  .  .  .  .
17 .  .  .  .  .  .  .  .  .  .  .  .  .
18 .  .  .  .  .  .  .  .  .  .  .  .  .
19 .  .  .  .  .  .  .  .  .  .  .  .  .

Key:
. = Empty
W = Wall
F = Filled

What row would you like to start with?
```

In this screenshot we are demonstrating how we have changed the shapes that will be generated when the CreateShapes() method is called upon. We have decided to change our shapes to rectangles, for at least now. The program starts by initializing all the cells and will randomly choose to create 3-5 rectangles to generate on the board. We will then run through a for loop to define the size of the rectangles, then pick random points at which to generate the rectangles. Then finally the method will connect the points together top to bottom, then will connect the points left to right to complete it generation of the randomly placed rectangles.

## Eight-Way

```
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9
0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
6  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
7  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
8  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
9  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
10 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
11 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
12 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
13 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
14 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
15 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
16 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
17 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
18 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
19 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .

Key:
. = Empty
W = Wall
F = Filled
NNorth Checking at -1, 14 Out of bounds. Stop.
NENorth East Checking at -1, 15 Out of bounds. Stop.
EEEast Checking at 0, 15 |
```

In this code I have updated the FloodFill algorithm with a few new updates. The first is I added the ability for the algorithm to check all directions, even diagonal

(N,S,E,W,NE,SE,NW,SW). We have also updated the FloodFill algorithm to now save its current point on the grid whenever it approaches a wall and then will be able to jump over the wall (in whatever the chosen position is in the console) then it will jump over the wall and begin the FloodFill at the next empty cell.

1. What was challenging?

I would have to say that the most challenging thing for this project was working with the wall jumping logic. Trying to figure out the wall jumping behavior without causing an infinite loop was a bit tricky. Then they also tried to handle the diagonal jumps and make sure there were not out of bounds errors. I think it is currently ok but still will need to play with it more.

2. What did you learn?

In this project I would say we learned a ton about recursion and how to control recursive calls and avoid stack overflows. Also learned more about direction tracking and how to pass directional context via parameters instead of needing to rely on error prone stack traces.

3. How would you improve on the project?

Maybe try to replace the Thread.Sleep with timer-based system for tracking how long each direct calculation. I feel this may be able to help the time each fill calculation takes and maybe be able to make them more even, but not too sure.

4. How can you use what you have learned in a job?

With different algorithm designs we will have the ability to be able to apply recursive problem solving to do path finding or any grid-based task, like game development or GIS tools. The debugging skills we have been picking up would also be useful in structure logging and step through debugging for those more complex algorithms.

