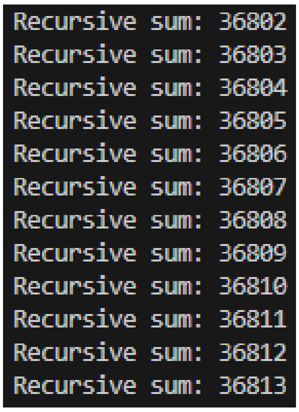
Assignment 1.1	
Using C++ for recursion	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 12/08/25**
Section: CpE21S4	**Date Submitted: 12/08/25**
Name(s): Francis Nikko I. Andoy	**Instructor: Jimlord Quejado**

6. Output

Recursive Sum:

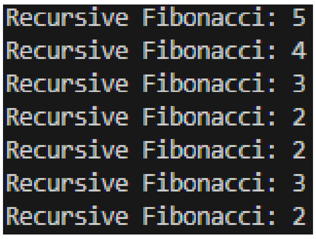


```
Recursive sum: 36802
Recursive sum: 36803
Recursive sum: 36804
Recursive sum: 36805
Recursive sum: 36806
Recursive sum: 36807
Recursive sum: 36808
Recursive sum: 36809
Recursive sum: 36810
Recursive sum: 36811
Recursive sum: 36812
Recursive sum: 36813
```

Observation:

Firstly, what is a recursive function, this function actually calls itself, during the execution. There is also a called base case if the condition is satisfied the program will stop. In the recursive sum I created a prototype function first, then I just made a random formula where the num which is the parameter in the function + 1. This will run infinitely, as we can see in the picture it reaches 36K since I didn't put any restriction or any condition where to stop.

Recursive Fibonacci:

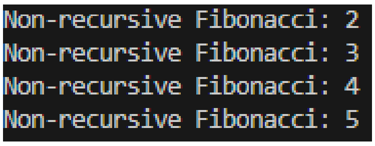


```
Recursive Fibonacci: 5
Recursive Fibonacci: 4
Recursive Fibonacci: 3
Recursive Fibonacci: 2
Recursive Fibonacci: 2
Recursive Fibonacci: 3
Recursive Fibonacci: 2
```

Observation:

In this recursive Fibonacci sequence, since we know what Fibonacci sequence is the formula and how to use it is also given.

Non recursive:



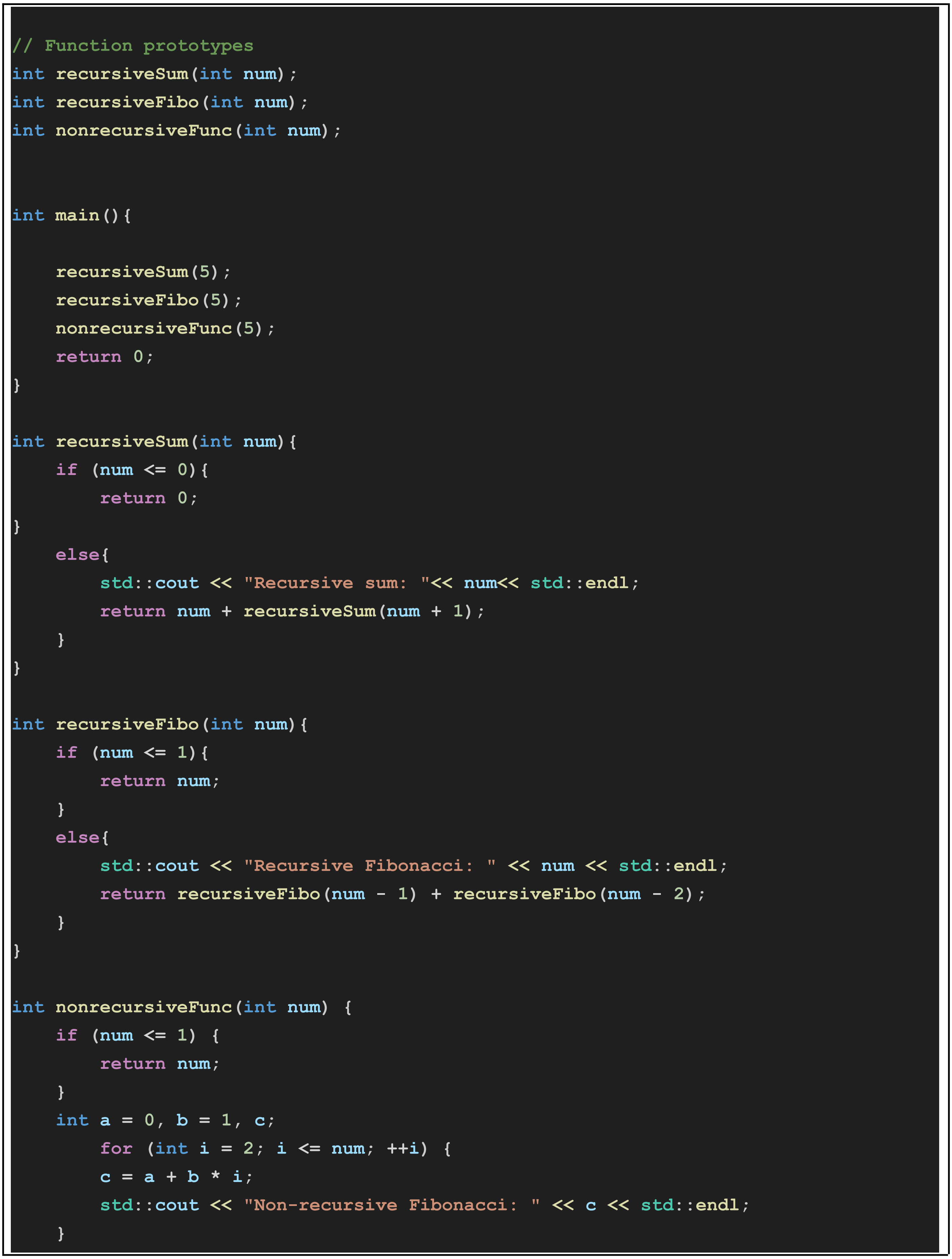
```
Non-recursive Fibonacci: 2
Non-recursive Fibonacci: 3
Non-recursive Fibonacci: 4
Non-recursive Fibonacci: 5
```

Firstly, what is a Non-recursive function, this is the complete opposite of the recursive function. This function doesn't call itself, it uses a loop where it can iterate the function.

7. Supplementary Activity

Code:

```
#include <iostream>
```



```cpp
// Function prototypes
int recursiveSum(int num);
int recursiveFibo(int num);
int nonrecursiveFunc(int num);


int main(){

    recursiveSum(5);
    recursiveFibo(5);
    nonrecursiveFunc(5);
    return 0;
}

int recursiveSum(int num){
    if (num <= 0){
        return 0;
}
    else{
        std::cout << "Recursive sum: "<< num<< std::endl;
        return num + recursiveSum(num + 1);
    }
}

int recursiveFibo(int num){
    if (num <= 1){
        return num;
    }
    else{
        std::cout << "Recursive Fibonacci: " << num << std::endl;
        return recursiveFibo(num - 1) + recursiveFibo(num - 2);
    }
}

int nonrecursiveFunc(int num) {
    if (num <= 1) {
        return num;
    }
    int a = 0, b = 1, c;
        for (int i = 2; i <= num; ++i) {
        c = a + b * i;
        std::cout << "Non-recursive Fibonacci: " << c << std::endl;
    }
    return b;
}
```

8. Conclusion:

In this activity, I learned that by using recursion we can break the problem into a smaller version of itself, and there is also something we can do instead of using a for loop, a function that calls itself and can run infinitely.
I also learned how to use the big O notation, and also on how we can differ these 2. A non recursive function uses a loop to iterate the given number it doesn't call itself like the recursive. Additionally, in the non recursive function we only used 1 for loop, we can solve the time complexity of this we can say that it was a O(n).

9. Assessment Rubric