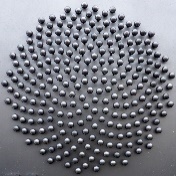
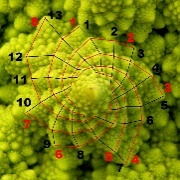
# CIS7 Unit 9 Lab: Recurrence in C++



[Fibonacci Ted Talk](https://www.ted.com/talks/arthur_benjamin_the_magic_of_fibonacci_numbers?language=en)

## Fibonacci sequence

The **Fibonacci sequence** exhibits a certain **numerical pattern which describe an amazing variety of phenomena, in mathematics and science, art and nature**. With mathematical ideas, Fibonacci sequence leads to the golden ratio, spirals and self- similar curves, have long been appreciated for their charm and beauty in the world of art and nature.

**The Fibonacci numbers are**

**0, 1, 1, 2, 3, 5, 8, 13, ... (add the last two to get the next)**

**0 + 1 = 1**

**1 + 1 = 2**

**1+ 2 = 3**

**2 + 3 = 5…**

By definition, the first two numbers in the Fibonacci sequence are either 1 and 1, or 0 and 1, depending on the chosen starting point of the sequence, and each subsequent number is the sum of the previous two.

***The sequence Fn of Fibonacci numbers is defined by the*** **recurrence relation** *(an equation that recursively defines a sequence or multidimensional array of values, once one or more initial terms are given)***:**

**Fn= Fn-1 + Fn-2**

with seed values: **F1 = 1 and F2 = 1**

or **F0 = 0, F1 = 1**

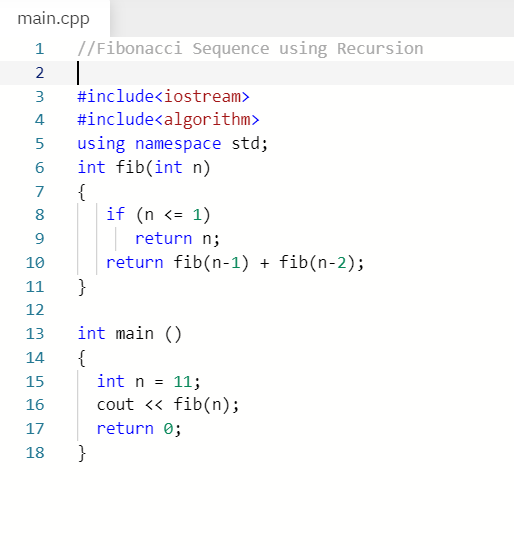
The Fibonacci sequence is named after Italian mathematician Leonardo of Pisa, known as Fibonacci; who introduced the sequence to Western European mathematics, although the sequence had been described earlier in Indian mathematics.

Applications of **Fibonacci numbers** include **computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure, and graphs called Fibonacci cubes** **used for interconnecting parallel and distributed systems**. They also appear in biological settings, such as branching in trees, phyllotaxis (the arrangement of leaves on a stem), the fruit sprouts of a pineapple, the flowering of an artichoke, an uncurling fern and the arrangement of a pine cone's bracts.

### Fibonacci

**Example 1**

The below C++ program illustrates Fibonacci sequence using recursion:



1. Write and run Example 1 program in IDE. Take a screen capture.

A screenshot of a computer

Description automatically generated

1. Verify if the program result is correct by showing the calculation of the nth value, given formula: **Fn= F n-1 + F n-2. Show calculation.**

**Fibonacci Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89**

**F11 = F10 + F9**

**F = 34 + 55 = 89**

1. Explain **how Fibonacci equation is integrated into the program** to derive the output.

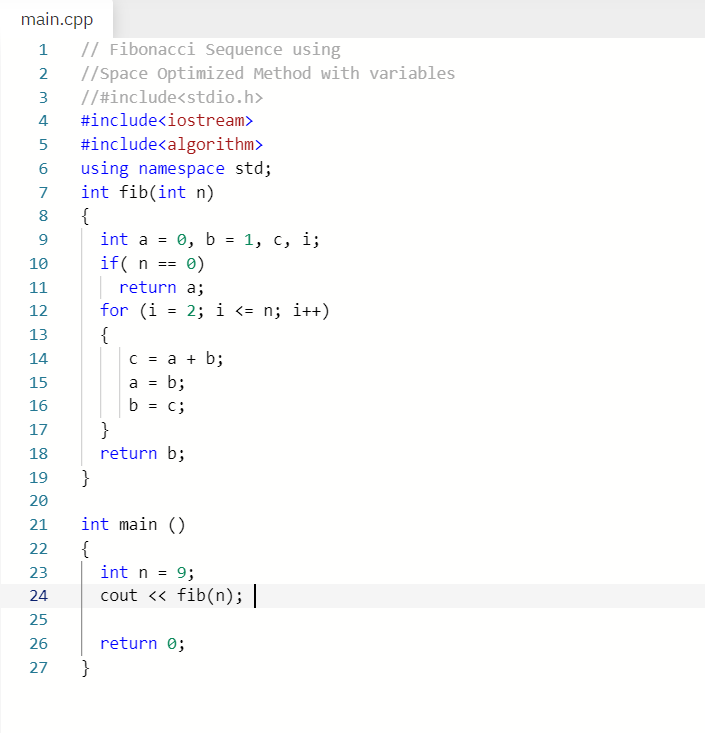
We recursively call the fib function twice in each call with n-1 and with n-2 and add the two each call. Once n <= 1 we return its value causing us to go back up call stack and eventually reach the first recursive call where we are returning Fn-1 + Fn-2

1. **Edit** the program or **create a program showing the calculation of the 18th Fibonacci term in the sequence**. Provide screen capture and result.

A screenshot of a computer

Description automatically generated

**Example 2:** C++ program using space optimization, variables, to determine nth value in Fibonacci sequence.



1. Take a screen capture and provide the result output.

A screenshot of a computer

Description automatically generated

1. Verify if the program result is correct by showing the calculation of the nth value, given formula: **Fn= F n-1 + F n-2. Show calculation.**

**F9 = F8 + F7**

**F9 = 13 + 21 = 34**

1. Explain how Fibonacci equation is integrated into the program to derive the output.

Two variables a and b are initialized to the first two numbers in the sequence. If n is zero then we just return 0. Otherwise we iterate through all the n values starting with n = 2 until we reach our desired n. Each iteration we set another variable c to the sum of a and b, a to b, and b to c. Then after all the iterations we return b.

1. Edit the program or create a program showing the **calculation of the** **14th term in the sequence**. Provide screen capture and result.

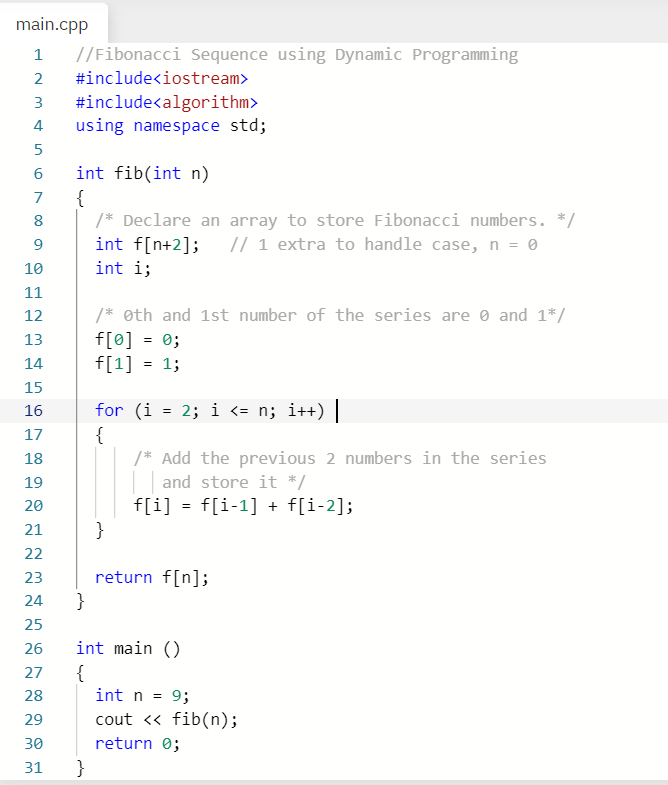
A screenshot of a computer

Description automatically generated

1. Test the program with different terms in the sequence. Provide a **comparison between Exercise 1 and 2 C++ programs (efficiency, use of variables, functionality).**

The program in exercise 2 is more efficient because it grows linearly while the program in exercise 1 grows exponentially since we are making 2 recursive calls each call. If we had used dynamic programming this would not have been the case. Furthermore, despite using extra variables in exercise 2 we are using less memory because in exercise 1 we need to store the additional stack frames for each recursive call.

**Example 3**: The following program calculates the 9th term of the Fibonacci sequence and store the result in an array.



1. Write and run the program. Take a screen capture and provide the result output.

A screenshot of a computer

Description automatically generated

1. Verify if the program result is correct by showing the calculation of the nth value, given formula: **Fn= F n-1 + F n-2. Show calculation.**

**F9 = F8 + F7**

**F9 = 13 + 21 = 34**

1. Explain how **Fibonacci equation is integrated into the program** to derive the output.

This program allocates an array the size of n + 2 to store the individual Fibonacci numbers. The first two numbers indexes 0 and 1 are initialized to values 0 and 1 respectively. Then we use a for loop starting at 2 running to n which says that the array at the index of our iterator i is equal to Fi-1 + Fi-2. Then we return the value in the array at index n.

1. Edit the program or create a program showing the **calculation of the 18th term** in the sequence. Provide screen capture and result.

A screenshot of a computer

Description automatically generated

1. Test the program for different terms in the sequence. Provide a **comparison between Exercise 1, 2 and 3 C++ programs** **(efficiency, use of variables, functionality).**

Exercise 3 and Exercise 2 both have the same time complexity with both being linear functions. While Exercise 1 is an exponential function due to 2 recursive calls in each call. Exercise 1 uses the least variables but takes up the most memory because of having to store stack frames. Exercise 3 uses only 2 variables but one is an array which depending on n could take up a large amount of memory. Exercise 2 uses the most variables but the least memory because it only stores 3 values as opposed to an array of values. Exercises 3 and 2 both use iteration to calculate the Fibonacci numbers while exercise 1 uses recursion to calculate the numbers.

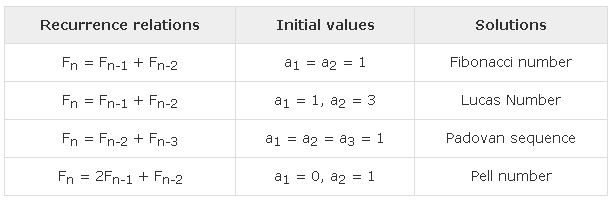
Recurrence

A **recurrence relation** is an **equation that recursively defines a sequence** **where the next term is a function of the previous terms** (Expressing Fn as some combination of Fi with I <n).

Linear Recurrence Relations

An equation that **recursively defines a sequence** or **multidimensional array of values**, **once one or more initial terms are given**: each further term of the sequence or array is defined as a function of the preceding terms.

Examples of linear recurrence equations:



## Recursive Function

Recursion: consists of a **function calling itself** in a potential **cycle of function calls.**

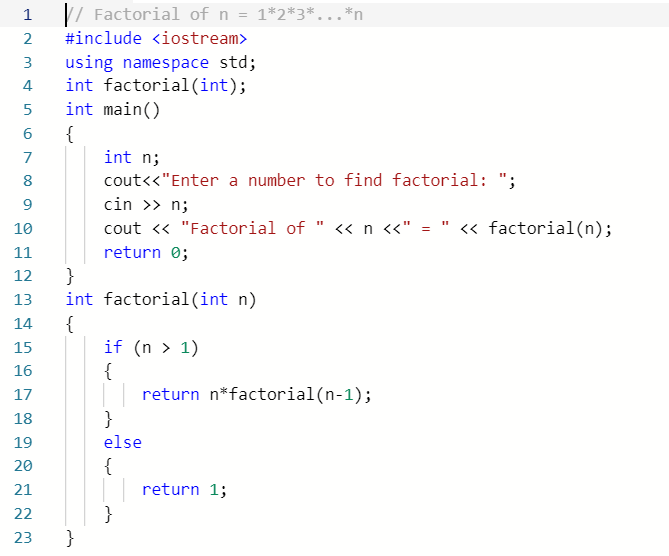
A **recursive function** is defined in terms of **base cases and recursive steps**.

**Base case** in a recursive algorithm is the case that is no longer recursive. Example: Factorial function.

In a base case, we compute the result immediately given the inputs to the function call.

In a recursive step, **we compute the result with the help of one or more recursive calls to this same function,** but with the inputs are reduced in size or complexity, closer to a base case.

**Example 4**: The following program demonstrates the use of recursive function in C++ to find factorial value of the user input:



1. Write and run the program. Take a screen capture and output.

A screenshot of a computer

Description automatically generated

1. Verify if the program result is correct by showing the calculation of the nth value.

F(10) = 1 \* 2 \* 3 \* 4 \* 5 \* 6 \* 7 \* 8 \* 9 \* 10 = 3628800

1. Explain how factorial algorithm is integrated into the program to derive the output.

When we call the function we first check the base case of if n is less than or equal to 1 if so we return 1. Otherwise, we recursively call the function with n-1 until we reach that base case such that each call we are returning n \* F(n-1). Once the base case is met we pop back up the stack until we reach the first call where N! is returned to be printed.

1. Edit the program or create a program showing the calculation for n = 15, n = 20. Is the output valid? Provide screen capture and result.

A screenshot of a computer

Description automatically generated

Both 15! and 20! Are not valid because they exceed the integer limit and start overflowing. In this case using logs would be more appropriate for large n values as no data type will be able to hold these values.

1. Explain the advantages and disadvantages of using recursive function for factorial calculation in programming.

The advantage of using recursive function for factorial calculation is the fact that factorial is inherently a recursive function making it very simple to implement recursively in our code. However, the drawback is that each recursive call needs to be pushed onto the call stack wasting large amount of memory especially if we are using large n values. Furthermore, unless we properly handle large n values by using tricks such as using log to calculate the factorial we will easily overflow integer limits for primitive data types.