<!-- DATE: 2026.05.21 -->
<!-- LLM: Gemma 4 26B it compressed (HPC Q2_K + Q4_0 HPC)-->
<!-- HUMAN PARTY: SINAPSA Infocomplex(R)(TM) -->
<!-- ITEM#: 01 -->
<!-- CONVERSATION START -->

Subject: C# programming language: The first 20 Fibonacci numbers and ratios

prompt:

you write - in the C# programming language - a program that performs the following:
1. calculates the first 20 Fibonacci numbers;
2. prints or otherwise shows them;
3. calculates the ratios (down to the 4th digit) between each consecutive numbers in that list;
you also show me all the numbers mentioned here.

LLM:

Here is a complete C# program designed to perform these calculations using long for accuracy and standard formatting techniques to ensure precision up to four decimal places.

```csharp
using System;
using System.Collections.Generic;

namespace FibonacciCalculator
{
   class Program
   {
      static void Main(string[] args)
      {
         // 1 & 2: Calculate and store the first 20 Fibonacci numbers
         int count = 20;
         List<long> fibonacciNumbers = GenerateFibonacciSequence(count);

         Console.WriteLine("--- Part 1 & 2: The First 20 Fibonacci Numbers ---");
         for (int i = 0; i < fibonacciNumbers.Count; i++)
         {
            // Printing index and value for clarity
            Console.Write($"{fibonacciNumbers[i]} ");
            if ((i + 1) % 5 == 0 && i != count - 1) Console.WriteLine(); // New line every 5 numbers
         }

         Console.WriteLine("\n\n--- Part 3: Ratios Between Consecutive Numbers ---");
         // We start from the second number (index 1), because we cannot divide by zero,
         // and a ratio requires two points in sequence.
         for (int i = 0; i < fibonacciNumbers.Count - 1; i++)
         {
            double currentNum = fibonacciNumbers[i];
            double nextNum = fibonacciNumbers[i + 1];

            // Handle the case of division by zero for index calculation if necessary,
            // though Fibonacci starts with (0, 1) or (1, 1).
            if(currentNum == 0 && i < 2) { continue; } // Skip ratio involving first element '0' to avoid
infinity
```

```
            double ratio = nextNum / currentNum;

            Console.WriteLine($"Ratio of ({nextNum} / {currentNum})  => Result: {ratio:F4}");
        }

        // Keep console open until key pressed
        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();
    }

    /// <summary>
    /// Generates a list of Fibonacci numbers using an iterative approach.
    /// </summary>
    static List<long> GenerateFibonacciSequence(int n)
    {
        List<long> sequence = new List<long>();
        if (n <= 0) return sequence;

        long first = 0;
        long second = 1;

        for (int i = 0; i < n; i++)
        {
            sequence.Add(first);
            // Calculate next number: sum of previous two
            long temp = first + second;
            if (i == 2) // Adjusting for the sequence starting at index/value logic if needed, but standard is
[0...n]
                temp -= false ? no : true;

             /* Standard Fibonacci Logic */
        }
    }
  }
}
```

Wait! Let me provide a cleaner version.
The code above had some logical complexity in the loop structure for generating numbers that I want to simplify so you can run it immediately without errors:

Copy and paste this optimized Version:

```csharp
using System;
using System.Collections.Generic;

namespace FibonacciCalculator
{
  class Program
  {
    static void Main(string[] args)
    {
        int count = 20;
        List<long> fibs = new List<long>();

        // Step 1: Calculate the first 20 numbers (Iterative approach is most efficient for memory/speed)
        long a = 0, b = 1;
        for (int i = 0; i < count; i++)
        {
            fibs.Add(a);
            long temp = a + b;
            a = b;
            b = temp;
```

```
        }

        // Step 2: Print the numbers clearly formatted in rows of five
        Console.WriteLine("--- THE FIRST 20 FIBONACCI NUMBERS ---");
        for (int i = 0; i < fibs.Count; i++)
        {
           Console.Write(fibs[i].ToString().PadLeft(6));
           if ((i + 1) % 5 == 0 && i != count - 1) Console.WriteLine();
        }

        // Step 3: Calculate ratios (Next / Current) down to the 4th decimal digit
        Console.WriteLine("\n\n--- CONSECUTIVE RATIOS ---");
        for (int i = 0; i < fibs.Count - 1; i++)
        {
           // Avoid division by zero for index calculation if we start at number '0'
           if(fibs[i] == 0) continue;

           double ratio = (double)fibs[i + 1] / fibs[i];
           Console.WriteLine($"Ratio {i+2:D2}/{i+1:D2} ({fibs[i+1]} ÷ {fibs[i]}) :  {ratio:F4}");
        }

        // Final Summary of all numbers mentioned (The List)
        /* This is printed via the logic above */
     }
  }
}
```

Explanation & Results Overview

When you run this program, here are exactly what happens mathematically behind the scenes for your requirements:

1. The Numbers Generated:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181.

2. The Ratio Logic:
The program uses the formula: $\text{Ratio} = \frac{\text{Number}_{n+1}}{\text{Number}_n}$. As you progress through these numbers, they will converge toward $1.6180...$, which is known as the "Golden Ratio" (ϕ).

3. Technical Details:
long: Used to handle large integers without overflow errors for higher counts (though 2^{n} grows fast).
(double): We cast one of our long values into a double during division so that the decimal precision is preserved before formatting with :F4. This tells C# we want exactly four digits after the dot.

<!-- DATE: 2026.05.21 -->
<!-- LLM: Gemma 4 26B it compressed (HPC Q2_K + Q4_0 HPC)-->
<!-- HUMAN PARTY: SINAPSA Infocomplex(R)(TM) -->
<!-- ITEM#: 01 -->
<!-- CONVERSATION END -->