. A continued fraction representation for the Golden Ratio ϕ is given by (as a sequence)

$$x_1 = 1 + \frac{1}{1}, \quad x_2 = 1 + \frac{1}{1 + \frac{1}{1}}, \quad x_3 = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1}}}, \quad \ldots\ldots \quad \phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}}$$

The sequence $\{x_n\}$ converges to $\phi = 1.6180339887498948482045868343656...$ as the number of iterations goes to infinity.

(a) Write a C or C++ function `double mygold(int n)`, that evaluates the n-th order iterated value of ϕ i.e. x_n, using the above algorithm. [4]

(b) The Golden Ratio may be used to compute the Fibonacci numbers using the relation below:

$$f_n = \frac{1}{2\phi - 1}\left(\phi^{n+1} - (1-\phi)^{n+1}\right), \qquad \text{for} \quad n = 2, 3, \ldots$$

This is an amazing equation. The right-hand side involves powers and quotients of irrational numbers, but the result is a sequence of integers.
Write a C or C++ function `double myfibo(n)` that evaluates the n-th order Fibonacci number using the above algorithm and approximating to the nearest integer value. [4]

Code Solution:

```
#include <iostream>

#include <cstdlib>

#include <cmath>


using namespace std;


double mygold(int n)

{

   double prod = 1.0;


   for(int i=0; i<n; i++) {

      prod = 1 + 1 / prod;

   }
```

```
    return prod;
}
double myfibo(int n)
{
    return (pow(mygold(50), n+1) - pow(1 - mygold(50), n+1)) / (2 * mygold(50) - 1);
}
int main()
{
    cout << "Phi = " << mygold(50) << endl;

    for(int n=1; n<=11; n++) {
        cout << n << ": " << myfibo(n) << endl;
    }

    return 0;
}
```