Core questions

Q2.1 We define the following sequence of natural numbers:

\[ c_{n+1} = \begin{cases} 
3c_n + 1 & \text{if } c_n \text{ is odd,} \\
\frac{c_n}{2} & \text{if } c_n \text{ is even,}
\end{cases} \]  

(1)

where we start the sequence from any integer \( c_0 > 0 \). Write a program that displays successive terms of this sequence, starting at either a value of \( c_0 \) that is either fixed or specified by the user (through std::cin).

If the sequence ever reaches 1, subsequent terms will be 4, 2, 1, 4, 2, ..., and the sequence becomes periodic. Your program should stop displaying the numbers in the sequence is once this occurs (i.e. when \( c_n = 1 \)).

You may find it useful to use the remainder operator, \( \% \): the expression \( a \% b \) evaluates to the remainder of \( a \) when divided by \( b \) (a number between 0 and \( b-1 \) – see C++ Primer pg. 151 for more details).

(Simple loops and condition statements)

Q2.2 Extend your program from the previous question so that it no longer displays the successive terms in the sequence, but instead calculates the smallest \( n \) such that \( c_n = 1 \) – the length of the sequence. Write a program that, for each \( c_0 \) from 1 to 100, displays the smallest \( n \) such that \( c_n = 1 \).

The Collatz conjecture states that for any initial number \( c_0 > 0 \), the sequence eventually ends up in this orbit. While the conjecture is still open, it has been checked numerically for \( c_0 < 10^{18} \) or thereabouts, so you can assume that the sequence will eventually reach 1!

(Simple loops and condition statements)

Q2.3 Write a program that displays the numbers from 1 to 100 – but replace multiples of three with the word 'Fizz' and multiples of five by the word 'Buzz'. Replace numbers that are multiples of both three and five with the word 'FizzBuzz'.

(Simple loops and conditional statements)

Q2.4 Write a function which takes the arguments \( a, b \) and \( c \), and displays the roots of the quadratic equation

\[ ax^2 + bx + c = 0. \]  

(2)

If the roots are complex, display the results in the form \( A \pm Bi \). Look in the documentation of the <cmath> library (http://en.cppreference.com/w/cpp/header/cmath) for a square-root function.

(Functions, square roots)

Q2.5 Write a program that takes a positive integer \( n \) specified by the user, and saves a text file containing the numbers 1 to \( n \), one on each line of the file.

(File output)

Q2.6 Write functions to evaluate the following expressions:

\[ f(\gamma) = 2 \tan^{-1} \left( \left( \frac{1 + \gamma}{1 - \gamma} \right)^{1/2} \sin \gamma \right), \quad g(\gamma) = \frac{1}{\sqrt{2\pi}} \log \left( \gamma^2 + 1 \right) e^{-\gamma^2}. \]
Evaluate and display the values of
\[ f(g(x)) \quad \text{and} \quad g(f(x + a) + g(b)). \]

where \( x = 0.15, a = 0.235 \) and \( b = 1.763 \). Recall that many of the standard mathematical functions are implemented in the `<cmath>` header: see http://en.cppreference.com/w/cpp/header/cmath. (standard library mathematical functions)

Q2.7 To calculate a root \( f(x) = 0 \) of a nonlinear function
\[ f(x) = 9x^4 - 42x^3 - 1040x^2 + 5082x - 5929, \]
we can use an iterative algorithm such as Newton's method. Starting from some initial guess of the location of the root \( x_0 \), we iterate
\[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \]

For a good enough initial guess, the sequence \( x_n \) converges to the root of \( f \).

We can implement this algorithm in C++ using the following code:

```cpp
#include <iostream>
#include <cmath> // for std::abs

double F(double x)
{
    // Note how we have rewritten the polynomial so that only
    // four multiplications are required.
    return (-(9.0*x - 42.0)*x - 1040.0)*x + 5082.0)*x - 5929.0;
}

double DerivF(double x)
{
    return ((36.0*x - 126.0)*x - 2080.0)*x + 5082.0;
}

double FindRoot(double x, int maxIter, double tol)
{
    for (int iter=0; iter<maxIter; iter++)
    {
        // Evaluate our function F
        double residual = F(x);

        // If |F(x)| < tol, we are close enough to the root, so return x
        if (std::abs(residual) < tol) return x;
```
// Perform Newton iteration step
x -= F(x)/DerivF(x);

// check if last iteration brought root close enough
if (std::abs(F(x)) < tol) return x;

// If we have not returned already, the iteration has not converged
// within maxIter steps. Print an error and return 0. We will look
// later at better ways of dealing with the error
std::cout << "Error: FindRoot did not converge after " << maxIter
    << " steps" << std::endl;
return 0;
}

int main()
{
    double x;
    std::cout << "Enter an initial guess: ";
    std::cin >> x;

    // Call FindRoot with 100 iterations, tolerance of 10^-6
    std::cout << FindRoot(x, 100, 1e-6);
    std::cout << std::endl;
    return 0;
}

Try running this program (available on the course web site as newton_example.cpp). You should find that your initial guesses converge to one of the roots, which are \( \pm 11 \) and the repeated root \( 7/3 \).

Modify this program to use the Secant Method,

\[ x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}. \]  

Note that this method does not require an implementation of \( f'(x) \), but does need two initial guesses, \( x_0 \) and \( x_1 \).

Q2.8 Write functions to calculate the Fibonacci numbers using

- 32-bit integer (\texttt{int}),
- 64-bit integer (\texttt{long long}),
- 32-bit floating point (\texttt{float}), and
- 64-bit floating point (\texttt{double})

data types\(^2\). In each case, describe what happens to the results for large enough \( n \) (you may need to look up to about \( n = 100 \)). To display all the digits of the floating-point variables, use \texttt{std::cout << std::fixed;}.

\(^1\)Can you find an initial guess that does not converge to one of these roots? For this choice of function \( f \) the Newton algorithm is very robust, but for some functions it is not easy to find an initial guess that \textit{does} converge.

\(^2\)\texttt{long long} is a C++11 feature, so may not be available on all compilers. The sizes of \texttt{int} and \texttt{long long} are not guaranteed to be 32 and 64 bits respectively, but this is often the case.
Q2.9 Write a program that displays two columns of numbers, \( x \) and \( x^2 \), where \( x = 0, 1/3, 2/3, \ldots, 2 \). The columns must be aligned, e.g.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.333333</td>
<td>0.111111</td>
</tr>
<tr>
<td>0.666667</td>
<td>0.444444</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.33333</td>
<td>1.77778</td>
</tr>
<tr>
<td>1.66667</td>
<td>2.77778</td>
</tr>
</tbody>
</table>

(Output stream width)

Further questions (harder, may require further reading)

Q2.10 Write a function to display a positive integer in base \( n \) (where \( n \) is 2–36, say). Use the characters a–z to represent the numbers 11 upwards. (Hint: for an integer \( i \) between 0 and 25 inclusive,

\[
\text{std::cout} \ll \text{static_cast<char>}(i + 97)
\]

displays a character from a to z.)

(recursion, conditional statements, radix conversion algorithm)

Q2.11 A twin prime is a pair of primes that differ by 2, for example (3, 5) or (41, 43). Write a program to count twin primes, and show that there are 35 twin prime pairs less than 1000. Then show that there are 440312 twin prime pairs less than \( 10^8 \).

Q2.12 Write a program which takes a positive integer specified by the user, and displays it in Roman numerals.

(User input, the modulus operator, conditional statements)

Q2.13 Write a program which takes a number in Roman numerals specified by the user, and displays it in decimal.