Core questions

Q1.1 Write, compile and run a C++ program which displays the text "Hello, World!", or any other string of your choice. If you use an IDE such as Microsoft Visual Studio to develop the code, try also running the code from the command line. (Compiling programs)

Q1.2 In the following piece of code, double-precision variables $a$ and $b$ are declared, and initialised with values 5 and 3 respectively.

```cpp
int main()
{
    double a = 5.0, b = 3.0;
}
```

Complete the program so that it displays the values of $a$, $b$ and their sum, for example

```
The value of 5 + 3 is 8.
```

You will want to add at least two (probably three) new statements to the program. (Displaying text and numbers, simple arithmetic)

Q1.3 Write a program which calculates the area and circumference of a circle of radius $\pi$.

Note that C++ does not have a definition of $\pi$ so you will have to define a variable to store $\pi = 3.14159 \ldots$ yourself. What type should this variable be – and how many digits of $\pi$ can it store?

Modify your program so that it displays the area and circumference for circles of radius $0, 1, 2, \ldots, 10$. (Simple floating-point arithmetic, loops)

Q1.4 Write a program to calculate the $n$th Fibonacci number. Recall that the Fibonacci numbers $f_n$ are defined by

\begin{align}
f_0 &= 1, \\
f_1 &= 1, \\
f_n &= f_{n-1} + f_{n-2} \quad \text{for} \quad n \geq 2.
\end{align}

(Hint: think carefully about how to perform one iteration of this sequence within a loop – you may need to define three variables, for the current, previous and next values of $f$) (loops, simple expressions)
**int and double arithmetic**

In lectures we covered the concept of operators (such as +, -, *, /), which can operate on variables. For the mathematical operations performed on double (fractional) variables, the results are usually as we would expect from mathematics. However, we can also perform these operations on int (integer) variables, and in this case, there are some subtleties to be aware of.

If both the operands of one of the mathematical operators (such as +, -, *, /) are of the same type (e.g. of type int), then the result of evaluating that operator will be of that same type. This means that when the division operator / is used on two int variables, it produces an integer result, even when the numerator is not exactly divisible by the denominator. When two operands of type int are divided, the result (if non-integer) will always be rounded towards zero to the next integer (e.g. 3.6 is rounded to 3, −3.6 is rounded to −3). Note that this not necessarily the closest integer.

This behaviour is particularly difficult to spot when the operands of the division operator are literals, i.e. if they are just a number, rather than the name of a variable. Numeric literals are considered to be of type int if they do not include a decimal point, so 3, −15, etc. are all of type int. To obtain a literal that is of type double, use a decimal point, e.g. 3.0, −15.0, 5.65. For double literals that are in fact integers, the zero after the decimal point is not needed, so 3. or 3.0 can be used to express a double variable with the value 3.

If either of the operands of an operator is a double, the other operand will be converted (or ‘promoted’) to the double type before the division takes place. That means that if either operand of the division operator / is a double, the result of the division will be of type double, and can be a fractional value.

A second subtle behaviour of int variables is in their maximum and minimum values. The largest number that an int can store is usually $2^{31} - 1 \approx 2 \times 10^9$. If an int variable is incremented by one beyond this, it ‘wraps’ to the smallest supported number $-(2^{31} - 1)$. This behaviour is usually undesirable, so integers should be kept within their maximum and minimum ranges. The unsigned data type is similar to int but stores natural numbers (non-negative integers). Although a unsigned variable cannot store negative numbers, it can store positive integers up to a larger maximum limit, $2^{32} - 1 \approx 4.2 \times 10^9$.

Q1.5 What value do you expect the variables a to j to be initialised to, and why?

```plaintext
double x = 5.0, y = 4.0;
int m = 5, n = 4;

double a = 4/3;
double b = 3/4;
double c = 1.0/2;
double d = 1/2.0;
double e = 1.0/2.0;
int f = 1.0/2.0;
double g = x/y;
double h = x/n;
double i = m/n;
```
double j = m/4;
double k = m/4.0;

Write a program to check whether your expectations were correct.

(Mixed int/double arithmetic and rounding)

Q1.6 The following code has several bugs (at least 5)

```cpp
#include <iostream>
#include <cstdlib>

// Program to calculate the sum:
// 1^2 + 2^2 + 3^2 + ... + 2000^2
//
// The result should be 2,668,667,000
int main()
{
    int n=2000;
    int sum=0;
    i = 1;
    while (i<n)
    {
        sum += i*i
        i++;
    }

    std::cout << "The sum of the first " << n << " squares is " << sum << std::endl;
    return 0;
}
```

Some of these are highlighted as errors by the compiler, but some may only become evident once the program has been run. Fix the errors.

Q1.7 In the statement

```cpp
int n = 3.8 + 3.8, m = -3.8 + -3.8;
```

what values are n and m initialised to? Explain why.

(Mixed int/double arithmetic and rounding)

Further questions

Q1.8 Recall that the binomial coefficient \( \binom{n}{k} \) is defined as

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!},
\]

where \( 0 \leq k \leq n \) and \! is the factorial. Write a program to display the value of \( \binom{n}{k} \). Does your program work for \( \binom{31}{5} \)? How about \( \binom{31}{16} \)?
Q1.9 Investigate the sequence,

\[ x_{n+1} = \alpha x_n (1 - x_n), \quad x_0 = 1/2, \]

where \(0 \leq x_n \leq 1\) and \(\alpha\) is a constant parameter, by displaying the first 30 or so terms of the sequence.

What happens for \(\alpha = 1.5\), \(\alpha = 3.2\), \(\alpha = 3.5\), \(\alpha = 3.627\)?

Instead of displaying the numeric value of each term in the sequence, try plotting a primitive graph, by displaying on each line a varying number of spaces (depending on the value of \(x_n\)) followed by a character of your choice.