1. Write a program to accept all required data in mm from the user and calculate the reading of a screw gauge in cm correct to three decimal places.
2. Generate the first n elements of the Fibonacci sequence. As a first case, do this using `while` looping, then re-write the program using `for` looping. Finally, after you have warmed up with basic loops, try a second case: write the program so that you accept a random number from the user and generate a Fibonacci sequence of as many elements as necessary so that the greatest element in the sequence is as close as possible to the number given by the user. (For example, in the first case, $n = 7$ must generate the sequence $0, 1, 1, 2, 3, 5, 8$ and, in the second case, the same $n = 7$ must generate the sequence $0, 1, 1, 2, 3, 5$.)

1. Write a program to accept an integer from the user and display the numbers before and after it. You are allowed to declare only one variable.
2. Program a calculator. It should be able to compute all arithmetic functions including squares, nth powers and roots. Ask the user for two inputs and then their desired operation; display the answer but do not end the program: find out if they want to add to or edit their previous inputs or manually close the program.

1. The following algorithm is known as 'Euclid's algorithm' and is used to find the greatest common divisor of two integers:

```
        PRINT Enter two integers
        INPUT a and b
ALPHA:  IF b = 0 THEN GOTO [GAMMA]
        IF a > b THEN GOTO [BETA]
        LET b = b − a
        GOTO ALPHA
BETA:   LET a = a − b
        GOTO [ALPHA]
GAMMA:  PRINT a
```

   Write a C program based on this algorithm.
2. Add, subtract, scalar multiply, and check for orthogonality, parallelism or obliqueness, two three-dimensional vectors given by the user.

1. The syntax `sizeof( )` takes a data type (`int`, `double` etc.) as its argument and is an identifier that will retrieve for you the allowed memory size of that data type. Print its value as a data type `%lu` and verify if the byte lengths given above for `int`, `char` etc. match those on your system. You may have to use the `limits` header file.
2. Use arrays and functions to ask the user the elements of a 3×2 matrix `A` and a 2×3 matrix `B`, then prove that `A`·`B` $\neq$ `B`·`A`. Remember to display any results/outputs as matrices.

   Next, write another program that computes the determinant of a matrix that the user provides.

1. Accept the lengths of the two shorter sides of a right-angled triangle from the user and do the following: (a) use Pythagoras's theorem ($c^2 = a^2 + b^2$) to find the hypotenuse, and (b) use the sine rule ($\frac{\sin A}{a} = \frac{\sin B}{b}$) to find the two unknown angles.
2. This is a lighthearted program to appreciate the use of arrays for purposes other than in a matrix. Store the mass, charge and spin properties (and any others you can think of) of electrons, protons and neutrons (and any other favourite particles you may have) in individual *global* arrays and write a program that asks the user up to three questions about the properties of these particles and, based on the inputs, guesses which particle it may be. Here are some particles: Graviton (0 mass, spin 2, unobserved, used for gravity), Photon (0 mass, spin 1, observed, used for electromagnetism), Higgs boson (125GeV mass, spin 0, observed), Electron neutrino (0.3eV mass, spin 1/2, observed, used for the atomic structure), Up quark (2 MeV, spin 1/2, detected, used for atomic structure).

1. Ask the user for the coördinates (coefficients only, to be entered as `(x, y, z)` into the program) of two vectors and (a) add them up, (b) subtract them, and (c) find their dot product.
2. The Maclaurin series of e^x is given by

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Compute this up to n terms as the user desires. You are *not* allowed to use the `math.h` function `exp(identifier)` anywhere in your program. (Use it only to verify your result.)

1. Ask the user for five numbers, whether integers or fractions, and write a program that outputs the largest and the smallest of these numbers.
2. Use arrays, functions and any other capabilities of a C program you have learnt so far to improve your earlier program coded for problem 2 in exercise 2 to determine the smallest and largest of five numbers provided by the user. You should now be able to make it considerably simpler than last time.

   Once that is done, improve your program so that, in addition to what it does now, it also displays the five numbers in ascending order and in descending order. Make sure your program works under these **use cases**: the numbers are already input in ascending or descending order; two or a few of the numbers are equal; the numbers are all equal.

1. Accept three random lengths from the user and see if they form a triangle. (Use the fact that for a triangle of sides `x`, `y` and `z`, all three conditions, `x + y > z` and `y + z > x` and `z + x > y` must be satisfied.) If they do not form a triangle, ask the user for three new values. Repeat until you have a triangle.
2. Add $x^{-3}\ \forall\ 1 \leq x \leq 10\ ;\ x \in \mathbb{Z}$ and display the result. If you are confident you can do this, start off with this modification: compute the same function for $1 \leq x \leq n$ in steps of m, where the user gives you both n and m.

1. Accept two arbitrary limits and list the prime numbers between, and including, these limits. (Remember to handle all possible scenarios: what if the user gives the upper limit first and then the lower? Or what if the user gives a composite number as one of the limits? Make sure your program is complete and, to start with, determines the order of the user's given limits as well as the first and last prime numbers to be printed properly.)
2. Produce a neat table of the three basic trigonometric functions for $0°$, $30°$, $45°$, $60°$, $90°$.

1. Use loops to accept `a` and `b` and determine a^b without using the `pow` command or calling `math.h` at all. If you got this right, let us complicate the problem slightly: use functions and loops to write a program that extracts the ones, tens, hundreds etc. digits from a number.

   Hint: What does `x/((int)pow(10,n)) % 10` do for, say, $x = 326$ and $n = 1$, and why?
2. Compute the `n`$^{\text{th}}$ root of a number given by the user. Let the user choose `n` too.

   HINT The `pow` operation works, but will using an integral expression such as `1/3` work? Recall that `1/3` evaluated as an integer is not `0.333` but simply `0`. Use `float` expressions like `1.0/3.0` instead.