Objects and classes

1. (5p) Overriding virtual functions. Write a GeometryShape class. It must have attributes width and height, and virtual methods calculatePerimeter to return the shape's perimeter (sum of all sides) and calculateArea to return the shape's area. Then write classes Rectangle and Triangle which inherit GeometryShape's attributes and methods and let you create Rectangles and Triangles with user-given width and height. In both classes, write an overriding function for calculatePerimeter and calculateArea. Calculate perimeters and areas for different rectangles and diameters in the main() function.
2. (5p) Operator overloading. Write a Vector3D class. It must have three double attributes that represent the elements of a 3D vector (for example: x, y and z). Overload the +, -, * and [] operators of Vector3D so that the plus and minus operators represent vector addition and subtraction, the asterisk operator represents dot product between vectors, and the square brackets operator returns the i'th element of the vector when called (for example "vector[1]"). Finally, overload the << operator of std::ostream so that you can print the vector with std::cout (for example, "std::cout << vector" would print "[1, 2, 3]"). Test the operators and methods in the main() function.
3. (5p) Pseudorandom number generation. Write a class for your own pseudorandom number generator. It should have a method to set a seed (initial value to generate random numbers from), and a method to generate and retrieve a pseudorandom floating-point value between a minimum and maximum value that are given as arguments.
   Instance two objects of the class with different seeds. For each object, generate 1000 pseudorandom numbers and calculate their means and standard deviations. Do your generators produce values that are normally or uniformly distributed, or something else?
   You are **not** allowed to use existing C/C++ functionality (such as rand, srand, random.h) to generate random numbers. Instead, you should implement your own. A relatively simple algorithm to implement is the linear congruential generator (https://en.wikipedia.org/wiki/Linear_congruential_generator). For calculating standard deviations and means, you can use C-style arrays to save multiple values under one variable name. For testing the distribution type of your generator, you can assume that a uniform distribution has a standard deviation of $(b - a)/sqrt(12)$, where b is the maximum value and a is the minimum value, and for a normal distribution, 68% of the generated values should be within one standard deviation of the mean of all values.