Hand in date: 19th November 2021

21WSC318 Embedded Systems Design and Implementation
Coursework 1 (30%)

There are two parts to the coursework.

1. Conversion from a structured C++ program to an object-oriented C++ program (20%)
2. Extending C++ classes so that they implement polymorphism (10%)

Submission for this Coursework should be in the form of a `.cpp` file called `cw1.cpp`.

Each task is to be performed individually. Task 2 relies on an attempt of task 1 but you should read this whole document to understand the scope of each task and to plan your work. The specifications for the tasks are shown below, initial code for the task can be found on the Learn server and the mark schemes are provided.

Task 1 - Conversion to an object-oriented C++ program

The aim of this task is to demonstrate an understanding of differences between structured and object-oriented approaches, particularly concentrating on the appropriate use of encapsulation.

The structured C++ programme `cw1.cpp` found on the Learn server is to be converted to an object-oriented C++ program. Its functionality should be unchanged following conversion.

Do not worry about overriding operators or stream inputs so that they work with your new classes yet, this will be done in task 2.

1. The program must use an object-oriented approach with more than one class and must contain only member functions (other than `main()` which should just create the first object).
2. Do not use global variables.
3. Do not worry about producing a carefully formatted user interface: a simple, neat interface is suitable.
4. Do not worry about making your input routines robust.
5. Document properly the code you add. Don't document every line of code - just the major blocks of code (see the marking scheme).
6. The structure of your code should be neat and easy to follow.
7. The filename `cw1.cpp` must be used for your source code.
8. I will test your code using compiler version 6 for MDK-ARM. You should check your code operates correctly for this compiler.

Task 2 Implementation of polymorphism

The aim of this task is to demonstrate familiarity with a number of features of object-oriented programming, principally the overloading of operators.

1. Following your conversion of the code to an object-oriented approach in Task 1, you will need to add further member functions or provide suitable functions to override operators. The functionality of each overloaded operator must be consistent with the corresponding functionality exhibited by inbuilt types.

    - Replace the functionality described by `L181 – L182` (in the original code) to print the information about the sensor with a single << operator to `cout` by overloading this operator with your class(es).

    - Replace the functionality described by `L319` and `L326` (in the original code) by overloading the + operator and defining functions that allow the operator to work with floats and your class(es).

2. The same guidelines (1-8) from the previous task apply to this task too.

Task 1 Specific:

1	Appropriate object-oriented design, ensuring that the program is suitably divided into objects.	/5
2	Proper use of encapsulation, principally to implement an object-oriented structure which ensures that objects are responsible for performing calculations on their own data.	/7
3	Appropriate use of other C++ features, such as correct use of call by value, call by reference and call by address, appropriate use of constructors, destructors and member functions. Where both exist, you should use C++ features rather than C features (note that characters and character arrays are part of both languages).	/3
4	Suitable layout and logical structure of the program as a whole, including neatness, suitable indentation in functions, loops and other structures, ordering of functions.	/3
5	Documentation of the program, to include title, purpose, limitations and any special requirements to compile the code. Documentation of the major structures (functions and classes) should be provided, including what they do (for example, 'Calculate the sum of the squares of two integers'), as well as explanation of arguments passed and values returned.	/2

Task 2 Specific:

1	Individual member functions operate as specified. This will include the assessment of the performances of the following. (a) the operations to add sensor values together through their classes, and an operation to add the class to a double floating point type to get a new double floating type. (b) the stream input and output operations to display the class values in an appropriate way.	/5
2	Do the operations you implement work correctly and efficiently? Tests will be added to the `main` function to test other cases.	/3
3	Structure of the additional features.	/1
4	Appropriate documentation related to the additional features.	/1

/30