4 Assignment P2 [Assignment ID: `classes`]

4.1 Preamble (Please Read Carefully)

Before starting work on this assignment, it is **critically important** that you **carefully** read Section 1 (titled "General Information") which starts on page 1 of this document.

4.2 Topics Covered

This assignment covers material primarily related to the following: classes, constructor, default constructor, copy constructor, move constructor, destructor, copy assignment operator, move assignment operator, operator overloading.

4.3 Problems

1. `DoubleVector3` CLASS. In this problem, you will develop a 3-dimensional real vector class called `DoubleVector3`. Download the file `DoubleVector3.hpp` from the course web site. This header file contains a complete specification of the interface for the `DoubleVector3` class and some supporting nonmember functions. Only the interface is given, the actual implementation is not. In this problem, you are to develop a complete implementation of all of the member functions of the `DoubleVector3` class and its supporting nonmember functions. Some important details regarding the interface for the class (and its supporting code) are given by comments in the header file provided. It is important to read these comments carefully in order to clearly understand exactly how the code must behave. The interface for the class (and its supporting code) must follow exactly the specifications given. Any declarations/definitions that are part of the interface for the `DoubleVector3` class must be placed in the header file `DoubleVector3.hpp`. The remainder of the source code for the `DoubleVector3` class should be placed in a file called `DoubleVector3.cpp`.
   Write a program called `testDoubleVector3` that thoroughly tests the above code. The source code for this program should be placed in the file `testDoubleVector3.cpp`. A simple (but incomplete) test program that you can use as a starting point for developing your own test program can be found on the course web site in a file called `testDoubleVector3.cpp`. Note that, for the purposes of marking, your code will be tested using a separate test program written by the course instructor (which will not be provided to you); that is, your own test code will not be used for this purpose. You must, however, include the source code for your test program (e.g., `testDoubleVector3.cpp`) in your assignment submission.
2. `Complex` CLASS. In this problem, you will develop a complex number class called `Complex`. The code should be structured as follows:
   (a) Any declarations/definitions that are part of the interface for the `Complex` class must be placed in a header file called `Complex.hpp`. The remainder of the source code for the `Complex` class (if any) should be placed in a file called `Complex.cpp`.
   (b) A test program called `testComplex` with all of its source code in the file `testComplex.cpp` should be provided that tests all of the functionality of the `Complex` class.

   (All file names must match exactly in terms of case with what is specified here.) The `Complex` class must satisfy the following requirements:
   (a) All of the code for your class must be in the global namespace. That is, do not actually place your code in any namespace of your own.
   (b) The code for your class should be self-contained, with only the following exceptions. You may make use of the class `std::ostream` for I/O (which is declared in the standard header file called `iostream`) and you may also use the functions `std::sqrt` and `std::atan2` (which are declared in the standard header file called `cmath`).
   (c) The `Complex` class must provide a constructor with the following properties. The constructor should take two **`double`** parameters, with the first and second being the real and imaginary parts (respectively) of the complex number to be created. Each parameter should have a default value of zero.
   (d) A copy constructor, move constructor, copy assignment operator, move assignment operator, and destructor should be provided. The code for these functions must be explicitly specified (i.e., do not simply let the compiler provide the code for them).

(e) The `Complex` class must provide member functions called `real` and `imag` that return a copy of the real and imaginary parts of the `Complex` object, respectively.

(f) The binary addition operator should be overloaded using a global function so that two `Complex` objects can be added. The binary subtraction, multiplication, and division operators should also be handled in the same way.

(g) The left-shift operator (i.e., `<<`) should be overloaded using a global function so that `Complex` objects can be output to a `std::ostream` using the usual syntax. That is, code like "`std::cout << z << "\n";`" should work as expected. A complex number should be output as a left parenthesis, followed by the real part, followed by a comma, followed by the imaginary part, followed by a right parenthesis, with no spaces in between. For example, writing $1-2j$ to a stream, would produce output like "`(1,-2)`".

(h) The equality operator (i.e., `==`) should be overloaded using a global function so that two `Complex` objects can be tested for equality.

(i) The non-equality operator (i.e., `!=`) should be overloaded using a global function so that two `Complex` objects can be tested for non-equality.

(j) A nonmember function called `abs` should be provided that takes as a parameter a `Complex` value z and returns a **`double`** corresponding to the magnitude of z.

(k) A nonmember function called `arg` should be provided that takes as a parameter a `Complex` value z and returns a **`double`** corresponding to the argument of z.

(l) A static member function called `getNumObjects` should be provided that takes no parameters and returns an **`int`** indicating the number of objects of type `Complex` currently in existence.

(m) Your code must be correct with respect to the use of the **`const`** qualifier.

(n) Your class cannot have any friend functions/classes.

(o) All of the function members of the class listed above must be public, unless explicitly indicated otherwise.

(p) All data members of your class must be private.

A simple (but incomplete) test program that you can use as a starting point for developing your own test program can be found on the course web site in a file called `testComplex.cpp`. Note that, for the purposes of marking, your code will be tested using a separate test program written by the course instructor (which will not be provided to you); that is, your own test code will not be used for this purpose. You must, however, include your test program in your assignment submission.

3. Consider the code associated with the `Complex` class from earlier in the assignment. Suppose that `w`, `x`, `y`, and `z` are objects of type `Complex` (i.e., as in "`Complex w, x, y, z;`"). For each expression given below: 1) fully parenthesize the expression in order to unambiguously show the order in which all operators are applied; and 2) rewrite the expression to explicitly show all function calls associated with operators. To do this problem, you will need to take into account the precedence and associativity of operators as well as whether each operator is overloaded as a member or nonmember function. For example, the expression `x - y + z` when fully parenthesized becomes `((x - y) + z)` and when translated into an expression that explicitly shows function calls for operators becomes **`operator`**`+(`**`operator`**`-(x, y), z)`.

   (a) `w = x + y + z`

   (b) `x = y = z`

   (c) `x * y == w / z`

   (d) `std::cout << x * y / z << "\n"`

4. Suppose that the function `myFunc` below utilizes the code associated with the `DoubleVector3` class from earlier in the assignment.

```
1  void myFunc()
2  {
3      DoubleVector3 u(1.0, 2.0, 3.0);
4      DoubleVector3 v(2.0 * u);
5      DoubleVector3 w;
6      if (u * u != 0.0) {
7          w = u + v;
8          std::cout << w << "\n";
```

```
9           }
10  }
```

For the purposes of what follows in this problem, assume that all compiler optimizations are disabled (including copy/move elision). Also, any function calls required to propagate the value of an object appearing in a return statement out of the function should be credited to (i.e., deemed part of) the function to which the return statement belongs.

(a) During the execution of the function `myFunc`, how many temporary objects of type `DoubleVector3` are created (considering only the code for `myFunc` itself, not the code for functions called by `myFunc`)? Rewrite the function `myFunc` to explicitly show these temporary objects by making them into named objects called `_tmp1`, `_tmp2`, and so on.

(b) During the execution of the function `myFunc`, how many calls are made (considering only the code for `myfunc` itself, not the code for functions called by `myFunc`) to the following member functions of the `DoubleVector3` class:
   i. the default constructor
   ii. the copy or move constructor
   iii. constructors other than the default and copy/move constructors
   iv. the destructor
   v. the copy or move assignment operator

(c) During the execution of the function `myFunc`, how many function calls are made in total, excluding the constructors, destructor, and copy and move assignment operators of the class `DoubleVector3` (considering only the code for `myFunc` itself, not the code for functions called by `myFunc`)?