AM3611: C++ for Scientific Computing
Assignment4: Classes

Due: 2 November

Note: Some student's with slightly older installations have compilers that revert to the 1998 C++ standard which can cause problems. For windows machines using WSL this sometimes leads to a warning that some syntax is incompatible with the 1998 version and it suggests using the compiler flag -std=c++11 to get rid of the warning (i.e. use the 2011 standard for c++ which is what I am following in lectures). However, is seems some older apple computers sometimes just give an error message that is not very helpful even though it is an issue with using the older standard. I would recommend using the -std=c++11 flag generally to ensure that you are using the more modern 2011 (or later) standard rather than the outdated 1998 version.

Problem 1: We created an IntArray class in the notes that illustrated the concept of RAII. The code for IntArray is posted on OWL in the folder PS4Code, IntArray.cpp. This problem is going to revisit problem 3.3 from the previous assignment so make sure you look at the feedback on your assignment as well as the comments posted in the announcement on OWL for assignment 2. Failure to take into account the feedback and comments on the previous assignment, even if you received full marks for your solution there, will be severely penalized when this problem is graded.

a) Using IntArray as a template, create a new class `DoubArray` that dynamically creates a double array using the RAII paradigm. In addition to allocating the space, the constructor should initialize the array elements to zero (You can initialize a dynamically allocated array with uniform initialization and if you put no numbers in the brackets it is assumed all entries are zero, e.g. `array=new double[5] {};` will dynamically allocate an array of length 5 with all entries 0).

b) Add a member function `PNorm` that finds the p-norm of the array assuming it is a vector of values (review the notes at the end of Chapter 5 if you do not recall what the p-norm is). For a long vector this can be a costly procedure so it is worthwhile to save the result as a member variable `m_norm` and return this value if the calculation has been previously performed by an earlier function call. This necessitates another member variable `m_normflag` that should be true if the old norm value can be reused and false if it needs to be recalculated. These member variables should be initialized in the constructor and the value of `m_normflag` should be reset whenever the values of the array are changed (e.g. when the copy constructor is used or `setValue` is called) or whenever p is changed..

c) Add an operator overload for the `–` operator and the assignment operator `=` that computes the difference of two arrays as vectors (i.e. if `a`, `b`, and `c` are of the `DoubArray` class then a statement like `c=a-b;` should be possible once these two operators have been overloaded). (There are a couple of examples in the class notes of how to do this. If you start on the assignment early, you may wish to leave this part of the question until we get to operator overloading in class). Note that the assignment operator should reset the `m_normflag.`

d) Add operator overloads to the DoubArray class for the `[]` operator, providing both a version for const objects and non-const objects and providing a check on the validity of the index (using an assert statement).

e) Alter your code from problem 3.3 (assignment 2) to take into account any feedback you received and the posted comments (just alter your code, don't redo the entire question at this point). Also, change the differential equation to: $y' = -y\ \sin x$. Note that it is still easy to solve the resulting difference equation: $\frac{y_n - y_{n-1}}{h} = -y_n \sin(nh)$ for y_n. Also, note that the exact solution is now $y = e^{-1+\cos x}$. Further alter your code so that it uses the `DoubArray` class where the length is dictated by the number of gridpoints `N`. This should be the only array type used by your code. Rather than reading N from the command line, write a function `getN` to ask the user for this value. Your implementation of the implicit Euler's methods should also be turned into a function `ImpEuler` that accepts as arguments `N` and `y` where N is the number of grid points and `y` is a `DoubArray` of appropriate length passed by reference and whose member m_array[0] is set to the initial value 1.0.

f) To test your code we will make use of the functions you created for the `DoubArray` class. You need to ensure all parts of your code are tested. Your tests should also include:
Write another function to fill a `DoubArray` with the exact solution (the arguments should be the same as your `ImpEuler` function). Rather than use the `getN` function for this test, loop over values of N=10, 20, 30, …100 and in each iteration of the loop find an exact solution, the implicit Euler solution, the difference between the exact and Euler solution (using the overloaded – operator), and the norm of the difference (using the `PNorm` function with p=1 and p=2). You should output to a file the value of `h` and the norm of the difference (for both p=1 and 2) between the exact and Euler approximation and include a plot of the norm of the difference versus `h`. Comment on what you find (Does the difference appear to go to zero as h goes to zero? Is the plot linear? What does the answer to the previous two questions tell you about whether or not your code is working and/or how good Euler's methods is for solving this equation? If I told you that I wanted either measure of the error to be less than 10^{-4} could you now give me a reasonable estimate of what N would be needed to achieve this? Which measure (p=1 or 2) requires higher N?)

Problem 2: This problem will use the RAII concept for matrices. You are reminded that the code for IntArray is posted on OWL in the folder PS3Code, IntArray.cpp. This problem is going to revisit problem 5.5 and 5.6 from the previous assignment so make sure you look at the feedback on your assignment as well as the comments posted. Failure to take into account the feedback and comments on the previous assignment, even if you received full marks for your solution there, will be severely penalized when this problem is graded.

a) Using the IntArray class as a template, create a class `DoubMatrix` that dynamically allocates `n x m` matrices. (the number of rows and number of columns should be member variables). The matrices should be created as contiguous blocks of memory as discussed in the notes and lectures for Chapter 4, not as was done in the textbook.
b) Write friend overloads for the << operator to output a `DoubMatrix` or `DoubArray` in a nice form (it should start on a new line, have a new line for each row of the matrix, and each field of the matrix should have a fixed width so that the columns always line up).
c) Rework your solution for Problem 5.5 to take into account any feedback you received and to take arbitrary sized matrices of class `DoubMatrix`. This should be a friend function of the `DoubMatrix` class that overloads the `*` operator.
d) Rework your solution for Problem 5.6 to take into account any feedback you received and to take arbitrary sized matrices of class `DoubMatrix` and vectors of class `DoubArray` from part a) of problem 2 above. The vector-matrix and matrix-vector multiplications should be overloaded * operators that are friend functions of both `DoubMatrix` and `DoubArray`.

e) Overload the assignment operator = so that it will take compatible objects of type DoubArray and DoubMatrix (you should have done this for DoubArray in problem 1).

f) Add a new member function that returns the p'th power of a matrix (first checking if the matrix is square). This function should make use of your previously created overloaded * operator.

As part c) now involves multiple classes, you should separate your file into a main.cpp driver program that tests various functions, a DoubArray.cpp and DoubMatrix.cpp for the class implementations and two header files DoubArray.h and DoubMatrix.h. You should now be able to test your class using expressions such as

```
A = C*D;
std::cout << "A = " <<  A << std::endl;
std::cout << "C*D =" << C*D << std::endl;
v = A.Pow(4)*v;
u = A*A*A*A*v;
std::cout << "u = " << u << "v = " << v << std::endl;
```

<u>Problems 6.1, parts 1-6, 8</u>: Use the ComplexNumber.h, ComplexNumber.cpp, and main.cpp posted on OWL in the folder PS3Code/ComplexNumberEx as the base code for this question.

<u>part 7</u>: Instead of what is in the text, alter the function from the previous assignment that determined if a pixel was in/out of the Julia set to use the ComplexNumber class (the computation of a new iteration should now be one line, making use of the CalculatePower member and the CalculateModulus function should be used in the while condition).

<u>What to hand in for each problem</u>: A full solution (as defined in the earlier assignments) for all problems and code for each problem on OWL.