EEEE2065 - **Lab Assessment 4**
Complex Numbers and the Mandelbrot Set.

Total marks: 100

In this lab assessment you are to create `.cpp` programs that accomplish the tasks described in the problem statement below. This lab assessment is made available to the students from Moodle on Wednesday Dec 14th and is due back uploaded on Moodle by 5pm, Friday Dec 23rd. You must use the OOP programming paradigm and classes in this assessment. This assessment is worth 10% of your final grade.

Marking Rubric

- 0-25%: Codes developed but did not compile properly.
- 26-50%: Codes developed and compiled, but not achieving expected result, programs are buggy or inefficient.
- 51-75%: Codes developed and compiled with correct bug-free and efficient results but modularity/clarity/commenting practices are not well followed.
- 76-100%: Codes developed and compiled using taught best practice standards, with clear, well-commented code and modularity rules being demonstrated/followed.

Plagiarism statement: The codes you submit are to be developed by the student alone. Any codes which are **significantly** similar to each other, or to code found on the internet will be penalized as plagiarism and points will be deducted accordingly.

1 Problem statement overview.

In this lab assignment, your task is to write a C++ program that implements a complex number class. You will populate the complex number class with class functions that add, subtract, multiply, divide, square and take the absolute value of the complex numbers.

In the second part of the assignment, you will use your complex number class to compute whether or not each element of an array of complex numbers is inside or outside of a set called the Mandelbrot set. A number is in or out of the Mandelbrot set depending on whether the number converges or diverges when iterating the equation `z=` $\texttt{z}^2$`+c`. You will then plot the Mandelbrot set using ascii characters.

Put all functions with more than ≥ 3 code statements into the separate source code file rather than coding them into the class.

2 `Complex` number class

2.1 Defining the `Complex` class

Create a class called `Complex` that has as variables a pair of coordinates `x` and `y`. Make these of type `double` and `private:` Create the following standard functions for your class:

1. Accessor functions to return each of your variables
2. Mutator functions to set your two variables, individually and jointly.
3. Default constructor initializes values to zero.
4. Parameterized constructor initializes the values to given parameters, use the mutator to do the setting.
5. Printing function. Overload this function to print both with and without an input variable name.

You also want to be able to perform mathematics, so write functions:

1. `void add(Complex c)` : adds the coordinate `c` to the internal coordinate.
2. `void sub(Complex c)` : subtracts the coordinate `c` from the internal coordinate.
3. `void mult(Complex c)` : multiplies the complex number `c` with the internal complex number
4. `void div(Complex c)` : multiplies the inverse of complex number $\texttt{c}^{-1}$ with the internal complex number
5. `void invert()` : inverts the internal complex number.

6. `void square()`: squares the internal complex number

7. `double abs()`: returns the absolute value of the internal complex number.

Then create the corresponding overloaded operators. Note that these functions return type `Complex`.

1. `Complex operator + (Complex c)` : the equivalent overloaded `+` operator, have it call the `add()` function on an appropriate variable to do the job.

2. `Complex operator - (Complex c)` : the equivalent overloaded `-` operator, have it call the `sub()` function on an appropriate variable to do the job.

3. `Complex operator * (Complex c)` : the equivalent overloaded multiplication `*` operator, that multiplies the two numbers and returns the value. Use the `mult()` function on an appropriate variable to do the job.

4. `Complex operator / (Complex c)` : the equivalent overloaded division `/` operator, that divides the two complex numbers and returns the complex value. Use the `div()` function on an appropriate variable to do the job.

5. `Complex operator ^ (int n)` : an integer power operation that multiplies the internal complex number by itself `n` times and returns the value. Use the `mul()` function in an appropriate way to do the job.

Some of these functions may be longer than 3 lines and their definitions should thereby go into the `.cpp` file. Test your `Complex` class functions in a function `testComplexClass()` that you call from the `main()` as we have been doing in the practicals. Ensure all the above-described functionality is working properly in your test function.

2.2 Background on Complex numbers

In general, complex numbers are made up of 2 parts: the real part `x` and the imaginary part `y`. It can be written as $z = \mathtt{x}+\mathtt{iy}$ or `(x,y)` where $\mathtt{i} = \sqrt{-1}$ is the imaginary number. Addition of complex numbers can be done component-wise: $(\mathtt{x}_0+\mathtt{iy}_0)+(\mathtt{x}_1+\mathtt{iy}_1) = (\mathtt{x}_0+\mathtt{x}_1)+\mathtt{i}(\mathtt{y}_0+\mathtt{y}_1)$, as can subtraction: $(\mathtt{x}_0+\mathtt{iy}_0)-(\mathtt{x}_1+\mathtt{iy}_1) = (\mathtt{x}_0-\mathtt{x}_1)+\mathtt{i}(\mathtt{y}_0-\mathtt{y}_1)$. Multiplication of two complex numbers is implemented as:

$$(\mathtt{x}_0+\mathtt{iy}_0)(\mathtt{x}_1+\mathtt{iy}_1) = (\mathtt{x}_0\mathtt{x}_1-\mathtt{y}_0\mathtt{y}_1)+\mathtt{i}(\mathtt{x}_0\mathtt{y}_1+\mathtt{x}_1\mathtt{y}_0).$$

Complex division is implemented as:

$$\frac{(\mathtt{x}_0+\mathtt{iy}_0)}{(\mathtt{x}_1+\mathtt{iy}_1)} = \frac{(\mathtt{x}_0+\mathtt{iy}_0)(\mathtt{x}_1-\mathtt{iy}_1)}{(\mathtt{x}_1+\mathtt{iy}_1)(\mathtt{x}_1-\mathtt{iy}_1)} = \frac{(\mathtt{x}_0\mathtt{x}_1+\mathtt{y}_0\mathtt{y}_1)+\mathtt{i}(\mathtt{x}_1\mathtt{y}_0-\mathtt{x}_0\mathtt{y}_1)}{\mathtt{x}_1^2+\mathtt{y}_1^2}$$

In particular, the inverse of a complex number is given as:

$$\frac{1}{(\mathtt{x}_0 + \mathtt{iy}_0)} = \frac{(\mathtt{x}_0 - \mathtt{iy}_0)}{(\mathtt{x}_0 - \mathtt{iy}_0)(\mathtt{x}_0 + \mathtt{iy}_0)} = \frac{(\mathtt{x}_0 - \mathtt{iy}_0)}{\mathtt{x}_0^2 + \mathtt{y}_0^2}$$

The absolute value of a complex number is given as: $\text{abs}(\mathtt{x} + \mathtt{iy}) = \sqrt{\mathtt{x}^2 + \mathtt{y}^2}$. These complex number mathematics are to be implemented into your complex number derived class.

In general, while real numbers can be conceptualized as a point on the real-number line, complex numbers are conceptualized as a point on the complex-plane with the real component lying along the `x`-axis and the imaginary component lying along the `y`-axis. In the next part on the Mandelbrot set we will be testing complex numbers in the complex plane to see if they meet the criteria to fall within the Mandelbrot set.

3 The Mandelbrot Set (MBS)

The Mandelbrot set is one of the most well known fractals consisting of a set of complex numbers $\mathtt{x}+\mathtt{iy}$ in the complex-number plane. There is a specific rule for testing whether a complex number is included in Mandelbrot set or not that is described in Section 3.2.

3.1 Defining the `Mandelbrot` class

Define a `Mandelbrot` class in which you will test a large number of points in the complex plane to check whether or not each point falls into the Mandelbrot set. For this purpose, you have to define a set of points $(\mathtt{x}, \mathtt{y})$ with ranges defined by your Mandelbrot class. Thus your Mandelbrot class should start out with the following private fields:

- `double xMin, xStep, xMax; // The range of x-elements in the MBS`
- `double yMin, yStep, yMax; // The range of y-elements in the MBS`

that hold the minimum, step and maximum values for both the `x` and `y` ranges of your Mandelbrot set. Then the `x`-values to be tested are {`xMin, xMin+xStep,...,xMax`} and the `y`-values to be tested are {`yMin, yMin+yStep,...,yMax`}. You also need variables keeping track of the number of `x` and `y`-values that you have as this will define the size of the array used to hold the Mandelbrot set. Define the number of points in the `x` and `y` ranges:

- `int Nx, Ny; // Number of x and y elements in each dimension of the MBS`

You will need to write a function to compute the values of `Nx` and `Ny` from `xMin, xStep, xMax` and `yMin, yStep, yMax` respectively.

The third private field you will need to define is a 2D array holding values of 1 or 0: a 1 indicates the corresponding point is in the Mandelbrot set while 0 indicates the point is not in the set. The easiest way to implement this is as a 1D array indexed as 2D. Add a 1D array (indexed as 2D) that you will dynamically allocate, to your class:

- `int *MBS; // array representing the 2D MBS.`

When you initialize your Mandelbrot object with values for `xMin, xStep, xMax, yMin, yStep, yMax`, the constructor should subsequently calculate the `Nx` and `Ny` values and then initialize your array `int* MBS` to have `Nx*Ny` values using:
`MBS = new int[Nx*Ny];`
and you need free the corresponding memory with `delete[]MBS;` in the destructor.

Create the following:

1. A default constructor to initialize your parameters to some sane startup values. You can use `xMin=-2;xStep=.02;xMax=1;yMin=-1;yStep=.02;yMax=1;` as your default values. `Nx` and `Ny` should be calculated and the required memory should be allocated in `MBS`.
2. A parameterized constructor where the input parameters are taken from the user, calculates the MBS size and allocates the memory.
3. A destructor that frees the memory in `MBS`.
4. A function `void calcNxNy()` to compute the values of `Nx` and `Ny`
5. A function `void printRange()` to print out the range of `x`, `y`, `Nx` and `Ny` parameters for the user to validate correctness.
6. A function (not a constructor) `void setRange(double xMin, double xStep, double xMax, double yMin, double yStep, double yMax)` that takes the values of `xMin, xStep, xMax, yMin, yStep, yMax`, saves them and recomputes `Nx`, `Ny` and then frees the old memory and reallocates new memory for `MBS`. This function may be called if the user ever wants to change the size or range of the Mandelbrot set being explored.

In both constructors, the values of `Nx` and `Ny` should be computed and the memory to hold `Nx`×`Ny` `ints` should be dynamically allocated.

At this point you are ready to compute the Mandelbrot set.

3.2 Determining the Mandelbrot set

The Mandelbrot set is determined by the complex-number equation/iteration:

$$\mathtt{z} = \mathtt{z}^2 + c \tag{1}$$

Equation (1) is iterated (repeated) over and over again starting with $\mathtt{z}$=0, if the value of abs($\mathtt{z}$) diverges (grows $\rightarrow \infty$), then c is **NOT** in the Mandelbrot set. Whereas if `z` converges, or oscillates between several points without diverging, then the complex number c **IS** in the Mandelbrot set. To clarify this definition of the Mandelbrot set, let us check the complex number $c = 0$ as an example:

- $\mathtt{z}=0 \rightarrow \mathtt{z}=\mathtt{z}^2+c = 0$
- $\mathtt{z}=0 \rightarrow \mathtt{z}=\mathtt{z}^2+c = 0$

and no matter how many times we iterate, the value of `z` never diverges, and therefore $c = 0$ IS in the Mandelbrot set and the corresponding value of the `MBS` array should be set to 1. If we try with $c = 1$

- $\mathtt{z}=0 \rightarrow \mathtt{z}=0^2+c = 1$
- $\mathtt{z}=1 \rightarrow \mathtt{z}=1^2+c = 2$
- $\mathtt{z}=2 \rightarrow \mathtt{z}=2^2+c = 5$
- $\mathtt{z}=5 \rightarrow \mathtt{z}=5^2+c = 26$

and here we see that after a few iterations equation (1) diverges ($|z| \rightarrow \infty$)) for $c = 1$. Therefore $c = 1$ is NOT in the Mandelbrot set and the corresponding value of the `MBS` array should be set to 0. A complex number c is inside or outside of the Mandelbrot set depending on whether the iteration $\mathtt{z} = \mathtt{z}^2 + c$ converges or diverges. You must come up with the test for determining the convergence or divergence of a complex number c yourself using the `Complex` class functions you have defined above.

You are to write the class function `calcMBS()` (together with supporting sub-functions) that, for the given values of `xMin, xStep, xMax, yMin, yStep, yMax`, compute whether each point is or is not in the Mandelbrot set and populates the 2D `Nx*Ny` array `int *MBS` accordingly.

3.3 Printing the Mandelbrot set

After you have populated your array `int *MBS` with the `calcMBS()` function, you need to write the `printMBS()` function to print out the 2D array to get a picture that looks as shown in Figure 1

```
Mandelbrot set range:
x:[-2:0.02:1], Nx=151
y:[-1:0.02:1], Ny=101
In function calcMBS()
In function printMBS()
```

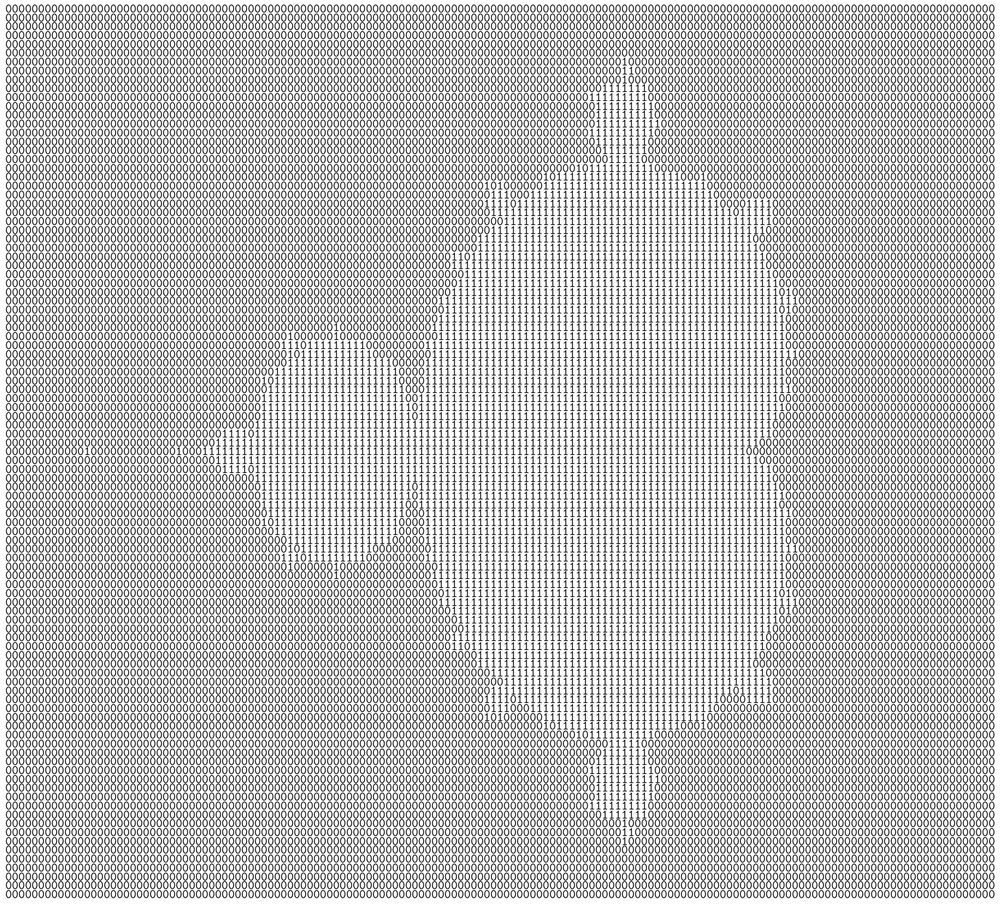


Figure 1: The output of the Mandelbrot set program with `x` ranging from -2 to +1 in steps of 0.02 and `y` ranging from -1 to +1 in steps of 0.02. A one indicates that complex number is part of the MBS, while a zero indicates it is not part of the MBS.

4 Report

Write up a report (5∼10 pages) detailing and documenting what you have done in this assignment. You should particularly highlight in your report, the codes that implement the above-described features. Have your report focus on the parts of the assignment that you had to create for yourself, such as:

1. How you implemented the operator overloading to return the object.
2. Details in how you wrote the function `calcNxNy()` function.
3. How you wrote the function to recalculate the the ranges and reallocate the memory for the MBS.
4. How you determined whether equation (1) converges or diverges

5. How you wrote the `calcMBS()` function to iterate over each point in the range and determined whether the point is inside or outside of the MBS. Explain the logic behind the design/creation of this function.

5 Submission and Evaluation

In order to standardize the evaluation of your work submit the following on Moodle by the due date/time:

1. `main.cpp` file: the main program. This program calls the high-level test functions for the `Complex` and `Mandelbrot` classes.

2. `CW4.cpp` file: contains the source code for all the functions used by your program to achieve the objectives. Be sure to use clear comments to delineate the different sections of your code file so that someone reading your code can clearly see where are the functions/operators for the `Complex` class, and where are the functions/operators for the `Mandelbrot` class.

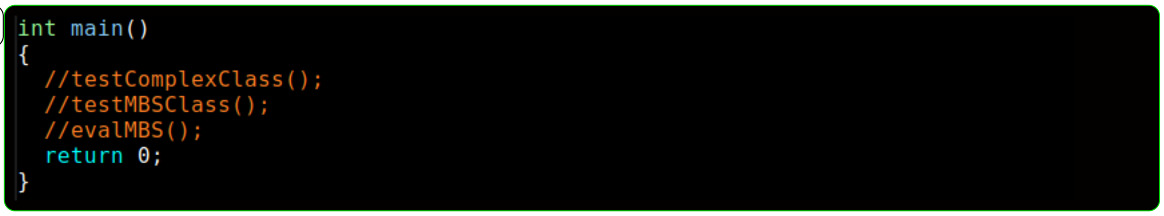
3. `CW4.h` file: the header file that holds both the function declarations and class definitions and that is included in both the `main.cpp` and `CW4.cpp` files. Functions/classes declared in `CW4.h` will be visible/callable in any file that `#include"CW4.h"`.

4. `report.pdf` file: the report.

If you have additional `.cpp` or `.h` files that you have written yourself and that are needed for the completion of the assignment you may also submit these. However, do not submit files that are created by the computer, including executables, object code, etc... You may submit the `.cbp` file too, although I will be compiling and testing your code with standardized functions that I write without it. In particular, one of the evaluation tests that will be carried out will be changing the range of `xMin, xStep, xMax, yMin, yStep, yMax` and seeing if your program can correctly recompute the MBS over a different range, so you should check that your program works for different ranges.

During evaluation, a set of `main.cpp` template codes will be created that call the functions and operators in your `Complex` class and check that each of your functions/operators are producing correct outputs for the given inputs. Your own default `main.cpp` that you submit should be setup to show how **you** tested and debugged your functions. An example main could look as follows:

1



```
int main()
{
  //testComplexClass();
  //testMBSClass();
  //evalMBS();
  return 0;
}
```

in which I can easily see and uncomment/test/check each part of your submission with appropriate testing functions written by you within each of the 3 main subfunctions. You should also be aware of the fact that I will be subjecting your classes and functions to standard tests that I will create for all students to check the correctness and completeness of your programs.

In addition, your programs will be tested for similarity with the programs of your classmates through a software to check for excessive duplicated code. Points will be deducted from both parties for duplicate code, and in severe situations, the case may be raised for additional disciplinary actions. The exception will be for duplicated code that is given to you in the practicals. But you are expected to comment your code to show your understanding of the given code, and to differentiate yourself from your classmates.