Programming in C++: Problem Set 2

Candidate No.253074

November 22, 2021

1 Intro

For these problems I have to demonstrate my ability to work with a range of features including classes, objects, pointers and functions.

Addressing this my first program will define a range of a circles variables in classes. By doing so I can create wanted functions to access said variables, as well as overload operators to allow the addition of circles as an example. I then plan to talk about classes and their variables features.

The second program looks at matrices and transposing them through use of addresses and pointers. To do so I first define a swapping function and then use this in a transpose function, allowing me to simply transpose the original matrix, saving memory. I also create a function to appropriately print matrices.

The final program is simulating an outbreak of a virus, I do so by modelling individuals within a class with the options of being alive, infected or immune using Boolean's to save memory. Therefore by using certain parameters for meeting people, causing infection, rate of recovery/immunity or death rate we can simulate the infection within a given population.

2 Circles

For this program (E1.cpp) I first defined and imported the Circle class (Circle.hpp). Said class holds a radius's magnitude and (x,y) coordinates for each Circle, as well as functions and operator overloads. Said functions all access each variable as a .get_() function (where _ represents radius, x coordinate or y coordinate) and the Circles area (πr^2), with $\pi = 3.14159265$ in this case; these all work as intended, returning the set values of created Circles.
The overloaded operators include a range of things:
1) The addition (+) of two circles to produce a third with the combined area of both in-between the two original circles.
2) The output («) of circles to display their variables and therefore be able to test with.
3) The comparison of size (>) to see which has a bigger area.

Using the overloaded '+' on circles A(3,2,1) and B(4,5,6) I returned circle C as C(5,3.5,3.5) which is the expected/wanted result. Circle C was displayed using both the .get_() functions and using the overloaded '«', with the other circles '«' was also used, all of which outputted the circle variables appropriately. I also created a Boolean function to see if two circles (X, Y) given areas were the same using only the overloaded '>' by seeing if either circles areas were larger than one another (X>Y, Y>X) and if not in both cases then they must be equal. This was used on circle B against A, B and C respectively, of which the results were 0, 1, 0; showing only [B, B] was found to be equal, as expected.

Note my class of 'Circles' used private member variables to hold key data. These are accessible or changeable through class functions, but if made public they could be accessible from outside the class object.

3 Transposing

For this program (E2.cpp) I first coded a swapping function swapr, which takes the addresses of two doubles and swaps their values by using a temporary variable. This is the used in a transpose function which transposes [3x3] matrices. It does so by looking at the matrix as ith row jth column and then swapping every [i,j] component with the [j,i] component on only one side of the diagonal. This is as if done to every component each would swap twice and give the original matrix.

To test the transpose function I then had to create a function to print it, printm. This prints each value of the row then ends the line and prints the next and so on. By printing the original matrix and then the transposed matrix, I can see if the code functions appropriately, said matrices have the following outputs:

$$O = \begin{bmatrix} 0.36 & 0.48 & -0.8 \\ -0.8 & 0.6 & 0.0 \\ 0.48 & 0.64 & 0.6 \end{bmatrix}.$$

$$O^T = \begin{bmatrix} 0.36 & -0.8 & 0.48 \\ 0.48 & 0.6 & 0.64 \\ -0.8 & 0.0 & 0.6 \end{bmatrix}$$

Therefore, seeing that these outputs show the matrix being transposed correctly, we can assume the transpose function and therefore the swapr function it uses are functioning correctly.

4 Infection

Looking first at this problem analytically, we know that if infected 12% recover and 1% die per day. Therefore 1 in 13 should die rather than survive, a percentage of 7.692% of those infected. Therefore if we find a similar ratio between the immunised and dead we have an accurate modelling for death and recovery.

For this program I first defined the Individual class in the Infection.hpp file. Said class holds 3 Boolean (to save memory usage) values, whether the modelled person, the individual, is infected, immune and alive; Individual(infected, immune, alive), is the format. It allows the inputting of all 3 values, only 2, 1 or none having the default values as Individual(1,0,1) for any amount of values inputted. The class has .get_() functions for all 3 variables allowing the evaluation of any given person, a .show() function to show all variables for a given person and .set(infected, immune, alive) function, allowing the change of state of a person. This allowed me to check the statuses of a given Individual and record the overall populace, as well as make changes to Individuals when needed and is therefore what is needed to continue to the main body of the program.

In the main program (E3_Func.cpp) we make use of the Individual class and a range of functions:

- *rnd(x)*, which creates a random integer from 0 to x, where x is a positive integer.
- *meeting(Individual A, Individual B)*, which sees if B is infected by A, checking to see if A is infected and the B is neither infected or immune, then using rnd(x) to determine if infection takes place
- *total(vector <Individual> inhabs, int A)*, finds the total number of infected, immune or alive if A equals 0, 1 or another number respectively from the vector *inhabs*

These functions are used to reduce repeated code, and if able made use of Boolean's where able to reduce memory usage.
These are then used to alter a vector of Individuals (Inhabitants) and find the total of each status for a given simulated day as they meet up, infect others, recover and die. The chance of someone infecting someone else whom isn't immune is 50%, the chance of recovery is 12% per day and the mortality rate is 1% per day. This allowed me to record the statuses of Individuals as this took place to plot the results below:

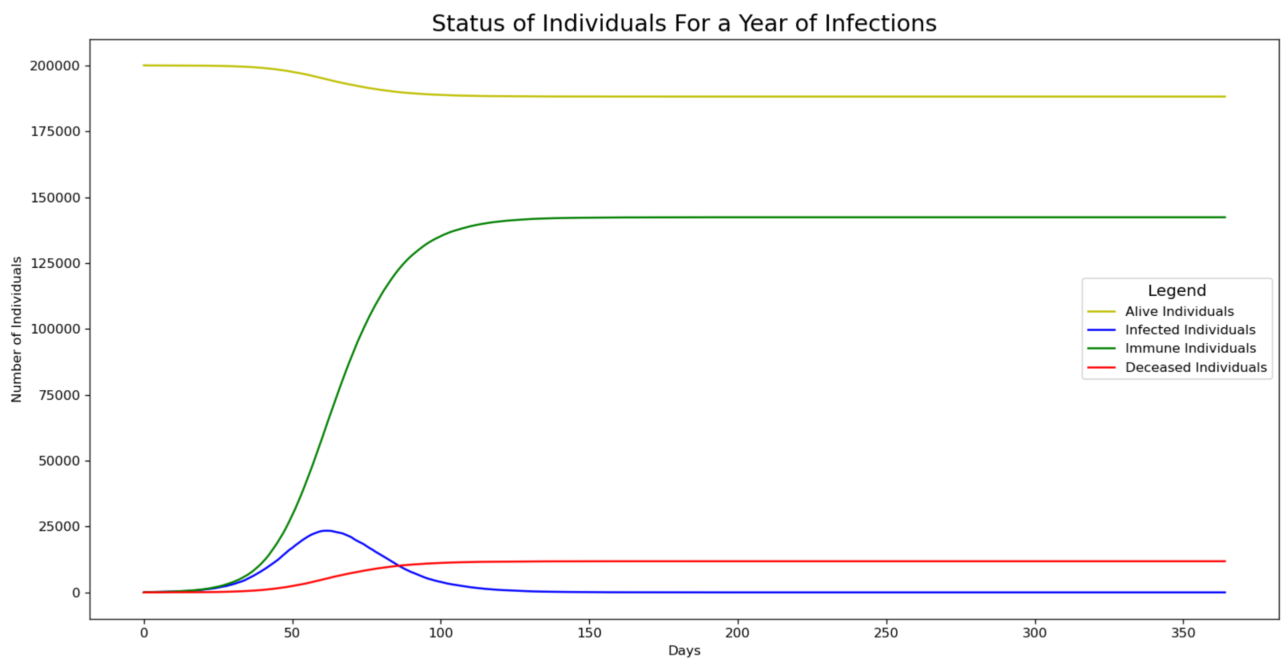

Figure 1: This figure depicts results of the infection simulation over the period of a year.

In figure 1, the peak of the infection is 23403 Individuals on the 62nd day with it ending (reaching zero infected) on the 196th day. This saw 142377 become immune and 188182 survive, meaning that 33987 Individuals were uninfected, 154195 Individuals caught the infection and of that 11818 died; this is a total of 7.66% of those whom were infected, which is a good approximation of our expected result. We also got a ratio of immunity of 75.7% at the end of the infection. From repeating this several times, comparing against the initial population size, I got an average/estimate for the outcome after a year of the alive and healthy population to be 94%, infected to be 0%, immune to be 71% and deceased to be 6%.

5 Conclusion

The Circle class worked as wanted and led to my program being effective at modelling the circles on a 2D plane as wanted. The Transposing program correctly accessed addresses and pointers to change the held information and as such could be used to transpose an array of said information. The Individual class was well thought through, holding the needed statuses while using Boolean's to reduce memory usage, which when used in vector form used less memory. Said class was then effectively used, through the use of functions, to simulate an infection within a population giving repeatable results.

Therefore, each program worked as intended using a range of features, such as classes, objects, pointers and functions, as appropriate. It can therefore be said, I was able to demonstrate my ability to use said features and do so effectively to solve a range of problems/programs.