Problem Sheet 2

Introduction to C++ programming

November 1, 2019

- Exercise 1 will let you practise the use of classes and objects.
- Exercise 2 will let you practise the use of pointers and references.
- Exercise 3 will introduce you to fractals.

1 Circle class

Write the declaration of a class called Circle with private members *radius*, *xpos* and *ypos* specifying the size and position of the circle in the x-y plane as floating point variables.

The class should contain the following public member functions:

- A constructor accepting as arguments radius, x-position and y-position, which are then assigned to the class members. Default values should be 0.0 for all members.
- Functions `getRadius()`, `getX()` and `getY()` returning the value of the corresponding member variables.
- A function `getArea()` returning the area $A = \pi r^2$ of the circle with radius r. The value for π should be defined as a global constant of type double with value 3.14159265 at the beginning of your code.
- A function `operator+(Circle c)`. This overloads the addition operator for circles. The function should return an object of type Circle with an area corresponding to the combined areas of the current circle and the circle c passed as an argument. The position (x, y) of the returned circle should be halfway between the current circle and the circle c. Use public member functions to access the parameters of c.
- A function `friend ostream& operator<<(ostream& os, Circle c)` returning the stream variable os. In the body of the function, radius, x- and y-position of the current circle should be written to os using the stream insertion operator <<. Format the output like this: ***radius = 0.0 at (x,y) = (0.0,0.0)***

Write a test program circle.cpp for the class Circle. In it, you should define two objects A and B of type Circle with radius 3.0 and 4.0, respectively. A is located at $(2.0, 1.0)$, while B is at position $(5.0, 6.0)$. Define another circle C obtained by adding circles A and B. The program should output the radius and coordinates of all three circles using the << operator of the class.

In addition, answer the following questions:

- How would you have to change your code to make the members *radius*, *xpos* and *ypos* accessible from outside the class object?
- Write the body of a member function for the class Circle with prototype
  ```
  bool Circle::operator>(Circle c);
  ```
  It should return `true` if the radius of the current circle is larger than that of circle c. Otherwise it should return `false`.
- Write down a boolean C++-expression which is true if and only if two objects A and B of type Circle are equal. Make only use of the operator ">" (for example as defined in b) in comparing A and B.

2 Matrix transpose

Write the following functions for variables of type double.

- Write a function `void swapr(..., ...)` swapping its two double arguments, passed by reference.
- Write a function `void transpose(...)` taking a 3×3 double array as an argument and transposing it, i.e.

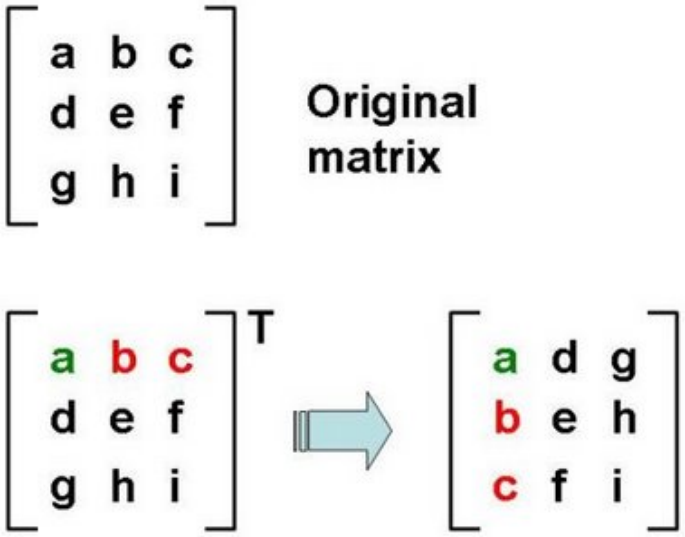
  $$A_{ij} \longrightarrow A_{ji} \quad \text{for} \quad i \neq j.$$

  Make use of the function swapr() above.
- Write a function `void printm(...)` taking a 3×3 double array as an argument and displaying it in three columns and three rows.

To test these functions, write a program transpose.cpp in which you define a 3×3 array M of type double. Initialize it to

$$\begin{pmatrix} 0.36 & 0.48 & -0.8 \\ -0.8 & 0.6 & 0.0 \\ 0.48 & 0.64 & 0.6 \end{pmatrix} \tag{1}$$

using comma separated lists. Using the functions above, the program should transpose and print the array M.



3 Mandelbrot

The Mandelbrot set is calculated as follows: For any point P in a 2-dimensional plane with coordinates (x_P, y_P) within the range

$$-2.2 \leq x_P \leq 1.0,$$

$$-1.2 \leq y_P \leq 1.2,$$

and a resolution (distance between two neighbouring points) of

$$dx = dy = 0.02,$$

apply the following transformations:

$$x_{n+1} = x_n^2 - y_n^2 + x_P,$$

$$y_{n+1} = 2x_n y_n + y_P,$$

with the initial values $x_0 = y_0 = 0$. Repeat these transformations as long as the point (x_{n+1}, y_{n+1}) is within the circle with radius $r = 2$ and center $(0, 0)$, i.e. fulfils the condition:

$$x^2 + y^2 < r^2.$$

Calculate the necessary number of iterations to leave the circle for each point (x_P, y_P). Please observe that some of these points are "stable" under the transformation: they will never leave the circle. Set $N = 100$ as maximal number of iterations. Print the output into a text file printing a char for every point (x_P, y_P): if the number of iterations n for a given point is equal to N, print the character '`o`'; otherwise, print the character '`-`'.

Task: Visualise the stable solution of the Mandelbrot set, and save it to a text file. It should look like a coarse version of this fractal image: