Scientific Computing

Assignment 6

Due: Friday, Nov. 10, 2017 at 17:00

Refer to week 1 lecture notes for instructions for submitting your assignment using the `hand-in` web form. In your submission include

- all required source code and
- any other files asked for in the question.

Put the files in a directory called `a6`. Do not include object or executable files.

For full marks document your work using *meaningful* comments.
Make sure to include

- your name,
- your student number, and
- the assignment number.

in each file you submit.

Also add comments where necessary to clearly label each solution. **Hand in only the source files and makefile, not the objects and executables.**

---

1. Copy `Chrono.cpp`, `Chrono.h` and `std_lib_facilities.h`[1] from the course website into your directory and `#include "Chrono.h", #include "std_lib_facilities.h"` in your assignment. Create a makefile that compiles the assignment with Chrono.o linked to assignment6.o. Write a function `Date prob1()` that asks the user to input a Date and print it to the screen, and returns the date.

2. Implement the `bool leapyear( int y)` function that returns `true` if `y` is a leapyear. Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2000 are. Run the function leapyear on the date entered by the user in problem 1 and print the result to the screen.

3. Make a short program `fltkPolygons.cpp` that writes "Drawing enclosed polygons" in an `fltk` Simple_window (you can start from ch12drill.cpp which is in the week8 examples folder, rename it, and remove all of the examples that are not needed). You will also need to download all of Bjarne's `fltk` interface code (Point.h, Window.h, Simpe_window.h, Graph.h, GUI.h, Window.cpp, Graph.cpp, and GUI.cpp).

4. Write a `main` function that uses se `argc` and `argv` to choose the number of sides on the polygon to use in the following problem (eg. `fltkPolygons.exe -N 15` would draw polygons with up to 15 sides.

5. Write a short program in `fltkPolygons.cpp` that draws a series of polygons that enclose each other. Start from a triangle at the center, enclosed by a square, enclosed by a pentagaon. The result should look like Fig. 1 without the circles (or optionally with the circles if you prefer). Also note that you can align the polygons at whatever angle you feel like. Make each polygon a unique colour, and draw up to N polygons (the number entered in the previous problem).

[1] `std_lib_facilities.h` has a number of includes from std libraries, and defines an error() function to call to exit the program in case of problems, and is used in the Date class and helper functions prepared by Bjarne Stroustrup

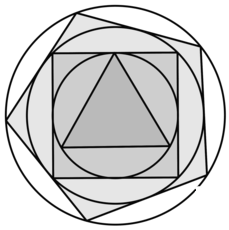

Figure 1: Triangle enclosed by square enclosed by pentagon (with circles drawn to help see where the polygons go). You don't need to shade the shapes in, or draw the circles.

Hints and formulae for figuring out coordintes of the polygon's corners

For some background on where the magic number 8.700036625... comes from, see http://mathworld.wolfram.com/PolygonCircumscribing.html.

If the largest outer circle enclosing all of the polygons has a radius R, then the radius R_{min} of the circle inside the triangle is $R_{min} = R/8.700036625$. The radius of the circle enclosing triangle has radius:

$$r_3 = R_{min}/\cos(\pi/3). \tag{1}$$

The radius of the N sided polygon is then:

$$r_N = R_{min} \prod_{n=3}^{N} \frac{1}{\cos(\pi/n)}. \tag{2}$$

Also notice that the verticies of the polygon should be equally separated in angle, so if we divide up 2π going all the way around the circle into N, we find that the vertices $n = 0, 1, ..., N-1$ of the N sided polygon should be at angles:

$$\theta_n = 2\pi n/N. \tag{3}$$

Therfore the polar coordinates of each vertex is (r_N, θ_n), from which the cartesian coordinates (x_n, y_n) of each corner of the N polygon can be found:

$$x_n = r_N \cos\theta_n = r_N \cos(2\pi n/N), \text{ and} \tag{4}$$

$$y_n = r_N \sin\theta_n = r_N \sin(2\pi n/N), \tag{5}$$

where $n = 0, 1, ..., N-1$.

For example if you want the outermost shape to have radius $R = 180$ pixels, and centered at $(x_c, y_c) = (200, 200)$ pixels. In this case the triangle would have vertices at radius $r_3 = 180/8.7000366625/\cos(\pi/3) = 41.37$, and vertices on the screen at (x'_n, y'_n):

n	$x'_n = 41.37\cos(2\pi n/3) + 200$	$y'_n = 41.37\sin(2\pi n/3) + 200$
0	241	200
1	179	235
2	179	164