Doc

Geometry (figury)

1 General Info

There are some techniques that can be used to achieve pseudo-OOP in C. Here standard classes are split in two parts:

- fields of a class are kept in a structure of the same name as a class would be named,
- methods are functions grouped by their names to match class name with certain prefixes and always take structure pointer as a parameter

Moreover abstraction in OOP can be simulated to some extent by adding additional field describing which exact object must be dealt with. In this way, as stated in task, structures are:

- struct geometry_point { double x; double y;};
- struct geometry_segment { geometry_point* start; geometry_point* end;};
- struct geometry_triangle { geometry_point* first; geometry_point* second; geometry_point* third; bool is_right;};

Whole task is implemented as a C library with only API visible, so forward-declaration of structures are used. To enable user to interact with structures some simple constructors/destructors/getters are also added.
This library also comes with a simple makefile with some rules written:

- 'make clean'- clean all generated code
- 'make test' - compile all existing tests
- 'make test_memcheck' - compile and run all existing tests using valgrind to check memory leaks

2 Functions implementations background

2.1 Moving by vector

Moving points on a coordinate plane by a vector with initial point in (0,0) is a simple task - one should just add coordinates of terminal point of this vector to coordinates of the point. If one wants to move whole shape described by some finite amount of points then all points can be moved one by one.

2.2 Rotating through an angle

If given is an angle ϕ measured in radians and calculated counterclockwise then this calculation can be pretty simple too, assuming rotation around (0,0) just use trigonometric functions:

$$(x, y) \to (x\cos\phi - y\sin\phi, x\sin\phi + y\cos\phi)$$

If rotation around some specified point (x_0, y_0) is need, then moving plane by vector forth and back can be applied:

$$(x, y) \to ((x - x_0)\cos\phi - (y - y_0)\sin\phi + x_0, (x - x_0)\sin\phi + (y - y_0)\cos\phi + y_0)$$

Once again - if one wants to rotate whole shape described by some finite amount of points then all points can be rotated one by one.

2.3 Calculating distances

Distance between two points $A = (x_1, y_1)$ and $B = (x_2, y_2)$ in euclidean space can be calculated with equation

$$|AB| = \sqrt{x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Of course length of a line segment is basically a distance between two endpoints and perimeter of a triangle is a sum of all sides length

2.4 Point in the segment

Let's assume that given point C lies on a line AB. Then of course exists $t \in R$ such that $A + t * \vec{AC} = B$. From this equation we can first derive t examining x coordinates of given points and then check if this value is correct (if it is then C lies on AB):

$$(x_A, y_A) + t * (x_C - x_A, y_C - y_A) = (x_B, y_B)$$

$$x_A + t(x_C - x_A) = x_B,\ y_A + t(y_C - y_A) = y_B$$

$$t = \frac{x_B - x_A}{x_C - x_A}$$

However implicitly calculating this division here is slow and inacurate, so let's put it into y-equation unchanged:

$$\frac{x_B - x_A}{x_C - x_A} * (y_C - y_A) = y_B - y_A$$

$$(x_B - x_A)(y_C - y_A) = (y_B - y_A)(x_C - x_A)$$

And this formula is put into the code; if got values match, then 3 points are colinear and it is only needed to check whether $x_C \in [x_A, x_B]$ and $y_C \in [y_A, y_B]$ - if A, B and C are colinear and C lies inside rectangle defined by points A and B, then C lies on segment AB.

2.5 Parallel/perpendicular segments

With two segements given, $AB = \{(x_A, y_A), (x_B, y_B)\}, CD = \{(x_C, y_C), (x_D, y_D)\}$, one can determine if they are parallel/perpendicular by comparing linear cooefficients of lines those segments lie on. Parallel lines fullfill equation $a_{AB} = a_{CD}$ and perpendicular lines fullfill equation $a_{AB} * a_{CD} = -1$. Adding that it is know how to calculate linear equation for line going through two point on a plane, we can derive equations:

$$\frac{x_B - x_A}{y_B - y_A} = \frac{x_C - x_D}{y_C - y_D}$$

and

$$\frac{x_B - x_A}{y_B - y_A} * \frac{x_C - x_D}{y_C - y_D} = -1$$

However division is slow and innacurate then it is best to use them in different form:

$$(x_B - x_A)(y_C - y_D) = (x_C - x_D)(y_B - y_A)$$

and

$$(x_B - x_A)(x_C - x_D) = -(y_B - y_A)(y_C - y_D)$$

. And that is how those checks are implemented in functions

2.6 Intersection point of segments - TBD

Of course from mathematical point of view simple algebra does the stuff here, however after some transformations of basic equations some complicated fractions with multiple subtractions and multiplications emerge - that's because this time calculations isn't about checking whether some equations hold or not and it is needed to calculate exact values of intermediate values. So there is high need for some more reasearch to find well designed algorithm that won't be suffering from those inaccuracies.

2.7 Area of a triangle

If the triangle is right-angled then it's area can be calculated from equation $P_{\triangle} = \frac{a*b}{2}$ where a and b are cathetuses. We only need to determine which of the sides these are - this can be done by checking perpendicularity of sides, what is already implemented (triangle's sides are segments). Otherwise we can use Heron's equation that uses only lengths of the triangle's sides:

$$P_{\triangle} = \sqrt{p(p-a)(p-b)(p-c)}$$

where:

$$p = \frac{a+b+c}{2}$$

2.8 Disjoint triangles - TBD

One of the simplest ideas is to check if some pairs of sides of given triangles intersect and this can probably be done with some tweaks to algorithm as there is no need for exact value, so let's drop some calculations. Of course this needs to be done after implementing algorithm for segment intersection.

2.9 Hypothenuse in right-angled triangle

To determine which of the triangle's sides is hypothenuse one only need to know which two sides are perpendicular, what is already implemented (triangle's sides are segments). After that length of hypothenuse can be calculated from Pythagoras Theorem: $c = \sqrt{a^2 + b^2}$.

3 Some remarks

3.1 Computation accuracy

For now all these algorithms are implemented as they can be seen above - with all equations exact, however because it is needed to deal with floating-point arithmetic all those calculations may be inacurate and maybe it would be better to give those programms some margin of error. It would be best to do some tests and decide how much of this margin is best for this library to give best outputs. Another thing to correct/optimise ("Premature optmization is the root of all evil" Donald Knuth) are trigonometric functions and roots - sometimes they will be inevitable of course but still maybe in some cases there exist solutions to eliminate them and thus increase accuracy of computations.

3.2 Error signaling

There is little to no error signaling as it is often making things messy and unclear. Especially if user needs to check every function return or error variable pointer needs to be passes every time as a function parameter. Possible elegant solution to this problem may be adding some global error flag in .h file and changing it adequatly inside each function. In this way user is able to check for errors every time it is needed and on the other hand it doesn't really interfere with whole API structure. Even more flexible solution is to put this code in macros and define them basing on compilation type - whether user wants to debug his code or efficiency is a main factor.