Computer Systems (SS 2011)
Exercise 3: May 9, 2011

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

April 8, 2011

The exercise is to be submitted by the denoted deadline via the submission interface of the Moodle course as a single file in zip (`.zip`) or tarred gzip (`.tgz`) format which contains the following files:

- A PDF file `Exercise`*`Number`*`-`*`MatNr`*`.pdf` (where *Number* is the number of the exercise and *MatNr* is your “Matrikelnummer”) which consists of the following parts:
  1. A decent cover page with the title of the course, the number of the exercise, and the author of the solution (identified by name, Matrikelnummer and email address).
  2. For every source file, a listing in a *fixed width font*, e.g. `Courier`, (such that indentations are appropriately preserved) and an appropriate *font size* such that source code lines do not break.
  3. A description of all tests performed (copies of program inputs and program outputs) explicitly highlighting, if some test produces an unexpected result.
  4. Any additional explanation you would like to give. In particular, if your solution has unwanted problems or bugs, please document these explicitly (you will get more credit for such solutions).
- Each source file of your solution (no object files or executables).

Please obey the coding style recommendations posted on the course site.

Exercise 3: Regular Polygons

Take the class Polygon from the previous exercises and adapt its interface as follows.

```
class Polygon
{
public:
  // usual operations
  Polygon();
  virtual ~Polygon();
  Polygon(const Polygon& poly);
  Polygon& operator=(const Polygon& poly);

  // add point and get number of points
  void add(double x, double y);
  int getNumber();

  // draws the polygon
  virtual void draw(unsigned int color1 = 0, unsigned int color2 = 0);

  // move polygon by vector (x, y)
  void move(double x, double y);

  // rotate polygon around point (x, y) by angle a
  void rotate(double x, double y, double a);
};
```

1. Derive from Polygon a class

   ```
   class RegularPolygon: public Polygon
   {
   public:
     ... // usual operations
     RegularPolygon(int n, int m, double x, double y, double r, double a);

     // getNumber() delivers n
     double getOffset();  // delivers m
     double getCenterX(); // delivers x
     double getCenterY(); // delivers y
     double getRadius();  // delivers r
     double getAngle();   // delivers a

     // draws the polygon including the center point
     virtual void draw(unsigned int color1 = 0, unsigned int color2 = 0);
   }
   ```

   where the constructor creates a regular polygon[1] with n points $0, \ldots, n-1$. All points lie on the circle with center $\langle x, y \rangle$ and radius r; the line from the center

[1] http://en.wikipedia.org/wiki/Regular_polygon

to point 0 has angle a to the positive half of the horizontal axis. The polygon connects each point i to point $i + m \bmod n$. The class shall make use of the data representation of `Polygon`, i.e. the constructor of `RegularPolygon` must compute the individual points of the polygon and call `add()` correspondingly.

The function `draw()` draws the polygon (using `Polygon::draw()`) but in addition it also draws the center point of the polygon.

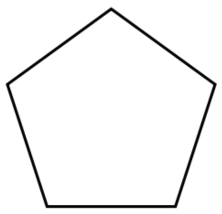
2. Derive from `RegularPolygon` a class

```
class SimplePolygon: public RegularPolygon
{
public:
  ... // usual operations
  SimplePolygon(int n, double x, double y, double r, double a);
}
```

which represents a simple regular polygon where each point i is connected to point $i + 1 \bmod n$.

3. Derive from `SimplePolygon` a class

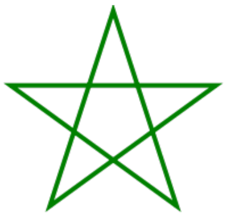
```
class Pentagon: public SimplePolygon
{
public:
  ...
  Pentagon(double x, double y, double r);
}
```



whose objects represent pentagons[2] (one point is vertically above the center).

4. Likewise, derive from `RegularPolygon` a class

```
class Pentagram: public RegularPolygon
{
public:
  ... // usual operations
  Pentagram(double x, double y, double r);
}
```



whose objects represent pentagrams[3] (one point is vertically above the center).

5. Finally write a class

```
class Picture
{
public:
  ... // usual operations
```

[2] http://en.wikipedia.org/wiki/Pentagon
[3] http://en.wikipedia.org/wiki/Pentagram

```
    Picture(double x, double y, double w, double h);
    void add(const Polygon& p);
    void draw(unsigned int color1 = 0, unsigned int color2 = 0);
    void move(double x, double y);
    void rotate(double x, double y, double a);
  }
```

A `Picture` object represents a rectangular picture which the constructor initializes with left upper point x, y, width w and height h such that its sides are horizontal/vertical. The picture consists of a boundary (a polygon representing the rectangle) and a set of polygons; the set is implemented by a linked list of `Polygon` objects (write a class `PolygonNode` to represent list nodes); initially this list is empty.

The function `add()` adds a *copy* of polygon p to the the front list (moving the copy by vector x, y in order to translate the relative polygon coordinates to absolute picture coordinates). The function `draw()` draws the picture (boundary and contents). The function `move()` moves the picture (boundary and contents) by a vector x, y, the function `rotate()` rotates the picture (boundary and contents) around point x, y by angle a.

Write a program that tests these classes, by creating a picture, populating it with pentagons and pentagrams, drawing the picture, moving and rotating the picture and displaying the results. Create a second picture as a copy of the first picture, display the copy, modify the copy, and show both the copy and the original (which must be not affected by the modification of the copy).