

Computer Systems (SS 2014)

Exercise 3: May 5, 2014

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

April 2, 2014

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 `ExerciseNumber-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

1. Write a class `Polygon` with the following public interface:

```
class Polygon
{
public:
    // create polygon in denoted color (default black)
    Polygon(unsigned int color = 0);
    virtual ~Polygon();
    Polygon(const Polygon& poly);

    // add point with relative coordinates (x,y) to polygon
    void add(double x, double y);

    // draws the polygon at absolute coordinates (x0,y0) scaled by factor f;
    // thus every point (x,y) is drawn at position (x0+x*f, y0+y*f)
    virtual void draw(double x0, double y0, double f);

    // returns pointer to copy of this polygon (calls copy constructor)
    virtual Polygon* clone();
};
```

The class internally maintains a linked list that holds the points of the polygon (objects of some user-defined class `Point`); for drawing the polygon, this list is to be traversed only *once*.

2. Derive from `Polygon` a class

```
class RegularPolygon: public Polygon
{
public:
    RegularPolygon(double x, double y, double r, int n, double a,
        unsigned int c = 0);
    RegularPolygon(const RegularPolygon& poly);
}
```

The constructor creates a convex regular polygon¹ with color c and n points $0, \dots, n - 1$ where each point i is connected to point $i + 1 \bmod n$. All points lie on the circle with center $\langle x, y \rangle$ and radius r ; the line from the center to point 0 has angle a to the positive half of the horizontal axis. The class shall make use of the data representation of `Polygon`, i.e. the constructor of `RegularPolygon` must compute the coordinates of the individual points of the polygon and call `add()` to add them to the polygon.

The function `draw()` is to be overwritten to draw the polygon (using the function `Polygon::draw()`) but in addition also the center point of the polygon (as a small

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

bullet). The function `clone()` is to be overwritten to call the copy constructor of `RegularPolygon`.

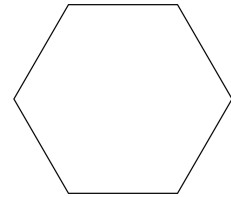
3. Derive from `RegularPolygon` a class with public interface

```
class Square: public RegularPolygon
{
public:
    Square(double x, double y, double r, unsigned int c = 0);
}
```

The constructor creates (by calling the constructor of `RegularPolygon`) a square with color c and horizontal/vertical sides whose bounding circle has center x, y and radius r .

4. Correspondingly derive from `RegularPolygon` a class with public interface

```
class Hexagon: public RegularPolygon
{
public:
    Hexagon(double x, double y, double r,
            unsigned int c = 0);
}
```



The constructor creates (by calling the constructor of `RegularPolygon`) a hexagon with color c whose bounding circle has center x, y and radius r such that two points are on the horizontal axis.

5. Finally write a class

```
class Picture
{
public:
    Picture(double x, double y, double w, double h, double f = 1.0);
    virtual ~Picture();

    void add(const Polygon& p);
    void draw();
}
```

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 plain rectangle) and a set of polygons; the set is implemented by an linked list of (pointers to) `Polygon` objects.

The function `add()` adds a *copy* of polygon p (use $p.clone()$) to the list. The function `draw()` draws the picture (boundary and contents); all polygons are scaled by the factor f used in the construction of the picture and shifted by the position x, y of the left upper corner of the picture.

Write a program that tests these classes, by creating a picture, populating it with squares and hexagons and drawing the picture. Please note that all polygons must be shown with their center point (this indicates that you have correctly overwritten `Polygon::clone()` and `Polygon::draw()` and called `Polygon::clone()`).