Object-Oriented Programming in C++ (SS 2022)
Exercise 3: May 12, 2022

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

February 2, 2022

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: Polygons

linked lists, inheritance

$\hookrightarrow$ class

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

```
class Polygon
{                  private should be added by us
public:
  // create polygon in denoted color (default black)
  Polygon(unsigned int color = 0);

  // copy constructor, copy assignment operator, destructor
  Polygon(Polygon &p);
  Polygon& operator=(Polygon &p);
  virtual ~Polygon();

  // create a heap-allocated copy of this polygon
  virtual Polygon* clone();

  // 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 = 0, double y0 = 0, double f = 1);
};
```

The class internally holds in a linked list the points of the polygon; for drawing the polygon, this list is to be traversed only *once*. Please note that `Polygon::clone()` has to be overwritten by every subclass of `Polygon` to preserve the dynamic type of the polygon (the method has to call the constructor of the respective class).

remember previous point and draw to the next

2. Derive from `Polygon` a class

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

all the constructor does is calling teh base class constructor, but first calculates the points accordingly

a = angle of the first point to the horizontal axis, radiants

The constructor creates a convex regular polygon[1] with color c and n points $0,\ldots,n-1$ 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 `Polygon::draw()`) but in addition also the center point of the polygon (as a small bullet); this center point has to be additionally stored in the `RegularPolygon` object.

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

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

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

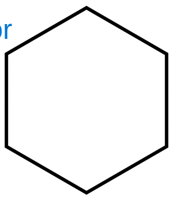
n=4, superclass constructor

The constructor creates (by calling the constructor of `RegularPolygon`) a square with color *c* bounding circle has center *x*, *y* and radius *r* such that the first point has angle *a* to the positive half of the horizontal axis.

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

n=6, superclass constructor

```
class Hexagon: public RegularPolygon
{
public:
  Hexagon(double x, double y, double r,
    double a = 0, 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 the first point has angle *a* to the positive half of the horizontal axis.

5. Finally write a class

```
class Picture
{
public:
  Picture();
  Picture(Picture &p);
  Picture& operator=(Picture &p);
  ~Picture();
  void add(Polygon &p);
  void draw(double x, double y, double w, double h, double f = 1.0);
}
```

collection of polygons
list class with nodes as polygons

pointer to the clones of polygons are stored

here, clone can be called

draws a rectangle of dimensions w and h and then iterrates through every polygon and draws them (clipping is allowed)

A `Picture` object represents a rectangular picture by a linked list of pointers to `Polygon` objects (the polygons passed in `add()` have to be cloned by `Polygon::clone()`; these copies have to be freed in the destructor of the class). The function `draw()` draws the picture by drawing a rectangular frame with left upper point *x*, *y*, width *w* and height *h*, such that its sides are horizontal/vertical. In the drawing, all polygons are scaled by the factor *f* and shifted by the position *x*, *y*.

Write a program that tests these classes, by creating a picture, populating it with irregular polygons, squares, hexagons, and other regular polygons, and drawing the picture. Create also a copy of the picture and draw it at a different position (please note that the copy of the picture needs to duplicate the individual polygons, no sharing is allowed). Please also note that all regular polygons must be shown with their center point (this indicates that you have correctly overwritten `Polygon::draw()`) and `Polygon::clone()`).