Computer Systems (SS 2012)
Exercise 3: May 7, 2012

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Wolfgang.Schreiner@risc.jku.at

April 19, 2012

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:
     // usual operations
     Polygon();
     virtual ~Polygon();
     Polygon(const Polygon& poly);

     // returns copy of this polygon (calls copy constructor)
     virtual Polygon copy();

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

     // draws the polygon (lines of color color1,
     // points as small bullets of color color2).
     virtual void draw(unsigned int color1 = 0, unsigned int color2 = 0);

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

   The class maintains an internal array that holds the points (objects of some user-defined class `Point`) $p_0, \ldots, p_n$ of the polygon to which new points may be added. If the array becomes full, a bigger array is allocated and the old array is disposed.

2. Derive from `Polygon` a class

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

   The constructor creates a convex regular polygon[1] with 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 individual points of the polygon and call `add()` correspondingly.

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

[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);
}
```

The constructor creates a square with horizontal/vertical sides whose bounding circle has center x, y and radius r.

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

```
class Pentagon: public RegularPolygon
{
public:
  Pentagon(double x, double y, double r);
}
```
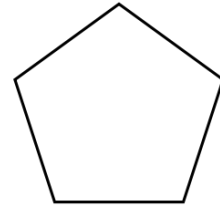
The constructor creates a pentagon[2] whose bounding circle has center x, y and radius r such that one point is vertically above the center.

5. Finally write a class

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

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

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 private class `PolygonNode` to represent list nodes); initially this list is empty.

The function `add()` adds a *copy* of polygon p (use $p.\text{copy}()$) to the front of the list (moving the copy by vector x, y 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.

Write a program that tests these classes, by creating a picture, populating it with squares and pentagons, drawing the picture, moving the picture and displaying the moved picture.

---

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