Computer Systems (SS 2011)
Exercise 4: May 23, 2011

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

May 2, 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 4: Generic Polygons

Generalize the solution of Exercise 1 such that point coordinates can have an arbitrary type C (which we assume to support the usual arithmetic operations like + and < and can be converted from/to int). In more detail:

1. Implement a class

   ```
   template<typename C> class Math
   {
   public:
     static bool equals(C c1, C c2);
     static int sign(C c);
   };
   ```

   which allows to compare two coordinates, determine the sign of two coordinates, and get a zero coordinate. Give the template class a reasonable default implementation (using the builtin operators == and <) but also construct a specialized template class Math<double> whose operators behave like in the class of Exercise 1 (i.e. use relative/absolute accuracies set by an additional function Math<double>::setAccuracy()).

2. Implement a class

   ```
   template<typename C> class Point
   {
   public:
     Point(C x = 0, C y = 0);
     C getX();
     C getY();
     void draw(unsigned int color=0, int radius=1);
     void draw(Point &p); // if needed in your solution
   };
   ```

3. Implement a class

   ```
   template<typename C> class Lines
   {
   public:
     static Point<C>* intersect
       (Point<C>& p0, Point<C>& p1, Point<C>& p2, Point<C>& p3,
         bool segment = true);
     static void drawIntersection
       (Point<C>& p0, Point<C> &p1, Point<C>& q0, Point<C>& q1,
          unsigned int color = 0);
   };
   ```

4. Implement a class

   ```
   template<typename C> class TPolygon
   ```

```
{
public:
  TPolygon();
  ~TPolygon();
  void add(C x, C y);
  void random(int n, int x, int y, int w, int h, int seed = 0);
  bool read(const char* filename);
  void draw(unsigned int color1 = 0, unsigned int color 2 = 0);
};
```

5. Implement two classes

   ```
   class Polygon: public TPolygon<double> { ... }
   class IntPolygon: public TPolygon<int> { ... }
   ```

   where the class `Polygon` must behave exactly as the class from Exercise 1. Test these two classes by constructing and and drawing for each class the polygon `poly1` from Exercise 1 and a randomly generated polygon.

Hint: in the intersection of lines, be careful in your use of the division operator in order to minimize truncation errors (which become problematic in the case $C =$ `int`). Rather than computing `a/b*c` you should therefore compute `a*c/b`.