Computer Systems (SS 2011)
Exercise 6: June 20, 2011

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

May 30, 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 6: Polygons with Container Parameters

Take the template classes `Math`, `Point`, and `Lines` developed in Exercise 4 and implement the following template classes for polygons:

```
// an abstract class for polygons with points of type Point<C>
template<typename C> class Polygon
{
public:
  virtual ~Polygon() { }

  // abstract functions
  virtual void add(C x, C y) = 0;
  vector< Point<C> > pointVector() = 0;
  set< Point<C> > pointSet() = 0;
  void draw(unsigned int color1 = 0, unsigned int color 2 = 0) = 0;

  // framework functions based on add()
  void random(int n, int x, int y, int w, int h, int seed = 0);
  bool read(const char* filename);
}

// a concrete class using Seq< Point<C> > for its internal representation
template<typename C, template<typename> class Seq> class SeqPolygon:
  public Polygon<C>
{
public:
  // add point x,y to polygon
  virtual void add(C x, C y);

  // get sequence/set of points (a copy of the internal sequence as a vector/set)
  vector< Point<C> > pointVector();
  set< Point<C> > pointSet();

  // draw the polygon
  void draw(unsigned int color1 = 0, unsigned int color 2 = 0);
};
```

Here `Seq` is assumed to be a class template that provides those operations that are common to all sequence containers of the C++ standard library (please note that `operator[]` is *not* among these operations). The class shall use objects of type `Seq< Point<C> >` for its internal representation and use *iterators* to process these objects (no duplicate of the point representation as an array/vector may be created for drawing the polygon).

For the implementation of `pointSet()` you have to specialize the template instance `less< Point<C> >` such that `less< Point<C> >(a,b)` returns `true` if point a occurs

before point b in the lexicographic ordering of point coordinates[1]; use the operations of class `Math` for performing the comparsons.

Then define template classes that use the C++ standard containers `vector` and `list` for their internal representation:

```
template<typename C> class VectorPolygon:
  public SeqPolygon<C, vector> {...};
template<typename C> class ListPolygon:
  public SeqPolygon<C, list> {...}:
```

Test the classes with the help of the template class `PolygonSequence` of Exercise 5 creating multiple polygons of types `VectorPolygon<double>` and `ListPolygon<double>`, storing all polygons in a sequence `seq`, and drawing the sequence by a call of `seq.draw()`. Also print the point sequence/set of some polygon with duplicate points (explicitly state the duplicates in the sequence).

[1] $(x_0, y_0) < (x_1, y_1) \Leftrightarrow x_0 < x_1 \vee (x_0 = x_1 \wedge y_0 < y_1)$.