Computer Systems (SS 2013)
Exercise 1: April 8, 2013

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

February 5, 2013

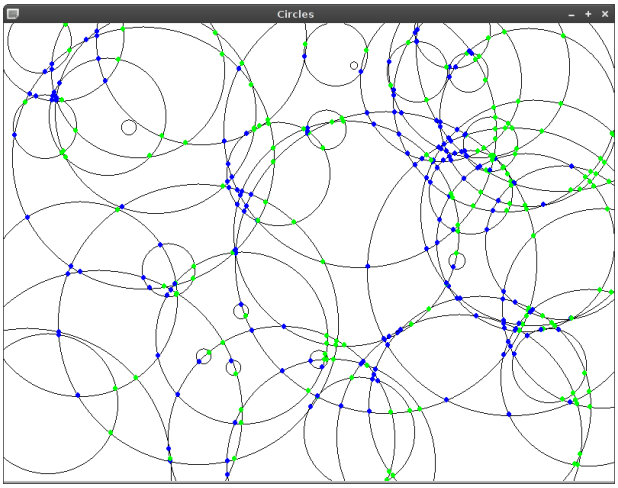
The exercise is to be submitted by the denoted deadline via the submission interface of the Moodle course as a single file in zip (`.zip`) or tarred gzip (`.tgz`) format which contains the following files:

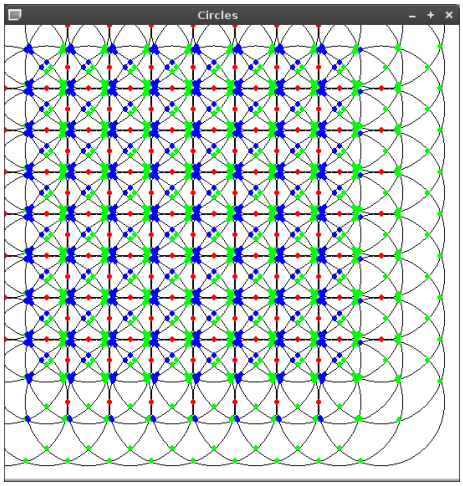
- A PDF file `Exercise`*`Number`*`-`*`MatNr`*`.pdf` (where *Number* is the number of the exercise and *MatNr* is your "Matrikelnummer") which consists of the following parts:
  1. A decent cover page with the title of the course, the number of the exercise, and the author of the solution (identified 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 1: Circle Intersections

Write a program that computes and visualizes the intersections of circles as shown below:





First recall how to intersect two circles with defining equations $(x-x_1)^2+(y-y_1)^2=r_1^2$ and $(x-x_2)^2+(y-y_2)^2=r_2^2$:

1. Subtract one equation from the other; the result is a linear equation in x and y which describes the line on which the intersection points of the circle (if any) reside.
2. Transform this equation into form $y=kx+d$ (or, alternatively, $x=yk+d$).
3. Perform the corresponding substitution for y (or x) in one of the circle equations yielding a quadratic equation $ax^2+bx+c=0$ (or $ay^2+by+c=0$).
4. Solve this equation to determine the intersection points.

The intersection of two circles may yield 0,1,2 intersection points (we treat the case where both circles coincide as if there were no intersection points).

Now write a C++ program `Circles` that performs the following tasks:

1. The program takes at most one argument from the command line. If there is no argument, this is an indication that 100 random circles are to be generated (use the random number generator `int rand()` defined in header `<cstdlib>`).
2. If there is an argument, this is the name of an input file that contains a sequence of lines

```
N
x1 y1 r1
x2 y2 r2
...
xN yN rN
```

where, for $1 \leq i \leq N$, (x_i, y_i) is the center point of a circle with radius r_i (all input values are integers, internally we compute with double precision floating point numbers). The program then generates and draws the indicated circles.

3. The program computes the intersection points of every circle with every other circle: if there is only one intersection, the point is depicted in red, if there are two intersections, one point is depicted in green, and the other one in blue (points are drawn as small filled circles).
4. The program prints the number of all intersection points to the standard output.

In detail, the program shall consist of the following components (you may freely introduce additional types and functions):

1. A class `Point` whose objects represent points and a class `Circle` whose objects represent circles (these objects are plain records, there is no need to augment the classes with constructors or member functions).
2. Two functions

   ```
   void drawPoint(const Point &p, unsigned int c=0);
   void drawCircle(const Circle &c, unsigned int c=0);
   ```

   which draw a point p respectively a circle c in color c (default black).
3. A function

   ```
   bool equals(double c1, double c2);
   ```

   which compares floating point numbers based on the values of two constants a, r (absolute and relative error)[1]: c_1 and c_2 are "equal", if $|c_1 - c_2| < a$ or else if $|(c_1 - c_2)/c_2| < r$ (for the second test, order c_1, c_2 such that $|c_1| < |c_2|$).
4. A function

   ```
   int intersectCircles(const Circle &c1, const Circle &c2,
                        Point &p1, Point &p2);
   ```

   which intersects two circles c_1 and c_2, returns the number n of intersection points (0,1, or 2), and, if $n \geq 1$, sets p_1 to the first point, and, if $n = 2$, sets p_2 to the second point. Compute in this function the coefficients k, d, a, b, c as described above and solve the resulting quadratic equation. Please consider all special cases (note division and square root computation).

   All floating point comparisons in this function are to be made with the help of the function `equals` explained above: in particular, a requirement for $f_1 < 0$ is that `equals(f1,0)` does not hold!

The program shall place all circles into an array `Circle* circles` which is dynamically allocated from the heap (and finally explicitly deallocated).

Test these components by at least the following tests:

[1] See http://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/

1. Read file `input1.txt` and draw the intersections in a window. The outcome must be as indicated in the left picture above.
2. Read file `input2.txt` and draw the intersections in a window. The outcome must be as indicated in the right picture above.
3. Generate 100 random circles and draw their intersections in a window.

Avoid code duplication (and recomputation of values) but make extensive use of auxiliary functions (and variables). Write for each class C a separate header file C`.h`. Use a separate file `Circles.cpp` for your test program.

Deliver the source code and screenshots of the three windows indicated above and the corresponding outputs of the program (the number of computed intersection points).