ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: B. Gärtner

Informatik für Mathematiker und Physiker HS12

Exercise Sheet 9

Submission deadline: 3.15pm - Tuesday 20th November, 2012

Course URL: http://www.ti.inf.ethz.ch/ew/Lehre/Info1_12/

Assignment 1 - (4 points)

Define a type Z_7 for computing with integers modulo 7. Mathematically, this corresponds to the finite ring $\mathbb{Z}_7 = \mathbb{Z}/7\mathbb{Z}$ of residue classes modulo 7.

For the type Z_7, implement addition and subtraction operators

```
// POST: return value is the sum of a and b
Z_7 operator+ (Z_7 a, Z_7 b);

// POST: return value is the difference of a and b
Z_7 operator- (Z_7 a, Z_7 b);
```

according to the following table (this table also defines subtraction: $x - y$ is the unique number $z \in \{0, \ldots, 6\}$ such that $x = y + z$).

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Assignment 2 - (4 points)

Consider the following program power.cpp. The function `linPower` calculates x^n using $n - 1$ multiplications. Let k be the number of digits of binary representation of n. In `power.cpp`, implement a function `logPower` that performs at most $2k$ multiplications. You can compare the two functions on the input $x = 2$ and $n = 10^5$.

```cpp
// Program: power.cpp
// This rogram computes x^n

#include <iostream>
#include <IFM/integer.h>

// Method that uses n-1 multiplications to compute x^n
// PRE: n >= 0
// POST: result is x^n
ifm::integer linPower(ifm::integer x, ifm::integer n) {
  if (n == 0)
    return 1;
  if (n == 1)
    return x;
  return x*linPower(x, n-1);
}

// Method that uses O(log n) multiplication to compute x^n
// PRE: n >= 0
// POST: result is x^n
ifm::integer logPower(ifm::integer x, ifm::integer n) {
  // Your code goes here ...
  return 0;
}

int main() {
  ifm::integer x;
  std::cout << "x =? ";
  std::cin >> x;
  ifm::integer n;
  std::cout << "n =? ";
  std::cin >> n;
  std::cout << "x^n = " << linExponent(x,n) << std::endl;
  return 0;
}
```

Hint: Look back at **Exercise 8**. Think, how you can generalize this approach. Also have a look at **Exercise 12**, part *a)*.

Assignment 3 - (4 points)

The C++ standard library also contains a type for computing with *complex numbers*. A complex number where both the real and the imaginary part are `double`s has type `std::complex<double>` (you need to `#include <complex>` in order to get this type). In order to get a a complex number with real part r and imaginary part i, you can use the expression

```
std::complex<double>(r,i); // r and i are of type double
```

Otherwise, complex numbers work as expected. All the standard operators (arithmetic, relational) and mathematical functions (`std::sqrt`, `std::abs`, `std::pow` ...) are available. The operators also work in mixed expressions where one operand is of type `std::complex<double>` and the other one of type `double`. Of course, you can also input and output complex numbers.

Here is the actual exercise. Implement the following function for solving quadratic equations over the complex numbers:

```
// POST: return value is the number of distinct complex solutions
//       of the quadratic equation ax^2 + bx + c = 0. If there
//       are infinitely many solutions (a=b=c=0), the return
//       value is -1. Otherwise, the return value is a number n
//       from {0,1,2}, and the solutions are written to s1,..,sn
int solve_quadratic_equation (std::complex<double> a,
             std::complex<double> b,
             std::complex<double> c,
             std::complex<double>& s1,
             std::complex<double>& s2);
```

Test your function in a program for at least the triples (a, b, c) from the set

$$\{(0,0,0),(0,0,2),(0,2,2),(2,2,2),(1,2,1),(i,1,1)\}.$$

Assignment 4 - (4 points)

Write programs that produce turtle graphics drawings for the following Lindenmayer systems (Σ, P, s).

b) $\Sigma = \{X, Y, +, -\}$, $s = Y$, and P given by

$$\begin{aligned} X &\mapsto Y + X + Y \\ Y &\mapsto X - Y - X. \end{aligned}$$

For the drawing, use rotation angle $\alpha = 60$ degrees and interpret *both* X and Y as "move one step forward".

c) Like b), but with the productions

$$\begin{aligned} X &\mapsto X + Y + +Y - X - -XX - Y + \\ Y &\mapsto -X + YY + +Y + X - -X - Y. \end{aligned}$$

Challenge - (8 points)

Exercise 135 from the lecture notes.