ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS13

Exercise Sheet 10

Submission deadline: 15:15 - Tuesday 26th November, 2013

Course URL: http://www.ti.inf.ethz.ch/ew/courses/Info1_13/

Assignment 1 – Skript-Aufgabe 136 (4 points)

Consider the following family of functions:

```
T foo (S i) { return ++i; }
```

with `T` being one of the types `int`, `int&` and `const int&`, and `S` being one of the types `int`, `const int`, `int&` and `const int&`. This defines 12 different functions, and all of those combinations are syntactically correct.

a) Find the combinations of `T` and `S` for which the resulting function definition is semantically valid, meaning, for example, that the constness of variables and references is respected. Semantical correctness also means that the compiler will accept the code, because we have already established syntactical correctness. Explain your answer.

b) Among the combinations found in a), find the combinations of `T` and `S` for which the resulting function definition is also valid during runtime, meaning that function calls always have well-defined value and effect; explain your answer.

c) For all combinations found in b), give precise postconditions for the corresponding function `foo`.

Assignment 2 – Skript-Aufgabe 138 (4 points)

We want to have a function that *normalizes* a rational number, i.e. transforms it into the unique representation in which numerator and denominator are relatively prime, and the denominator is positive. For example,

$$\frac{21}{-14} \quad \text{is normalized to} \quad \frac{-3}{2}.$$

There are two natural versions of this function:

```
// POST: r is normalized
void normalize (rational& r);

// POST: return value is the normalization of r
rational normalize (const rational& r);
```

Implement one of them, and argue why you have chosen it over the other one.

Hint: You may want to use the function `gcd` from Lecture 8 (Script Section 2.9), modified for arguments of type `int` (how does this modification look like?).

Assignment 3 – Skript-Aufgabe 131 (4 points)

a) Provide definitions for the following binary relational operators on the type `rational`. In doing this, try to reuse operators that are already defined.

```
// POST: return value is true if and only if a != b
bool operator!= (rational a, rational b);

// POST: return value is true if and only if a < b
bool operator< (rational a, rational b);

// POST: return value is true if and only if a <= b
bool operator<= (rational a, rational b);

// POST: return value is true if and only if a > b
bool operator> (rational a, rational b);

// POST: return value is true if and only if a >= b
bool operator>= (rational a, rational b);
```

On the website you find `rational.cpp` and `rational_test.cpp`. The file `rational.cpp` defines the struct `rational` and contains some of the operators that were discussed in the lecture. Add your solutions to this file. Then compile and run `rational_test.cpp` to test your functions on some examples.

b) Write a program `rational_sort.cpp` that reads n rational numbers from standard input into a vector. The number n is the first input, and then the program expects you to input another n values. After reading the n values, the program should sort them and output the sorted sequence. For example on input `3 6/7 1/2 1/4` the output should be `1/4 1/2 6/7`. You also find some larger sample inputs on the website.

Hint: Use the operators that you have defined for the type `rational`, especially `operator<<` and `operator>>`. If `operator<` is defined, `std::sort` can be used to sort the vector!

Assignment 4 – Skript-Aufgabe 141 (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

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

(As you can easily check yourself, the corresponding return values should be $\{-1,0,1,2,1,2\}$.

Challenge – Skript-Aufgabe 142 (cubic equations)