# Computer Systems (SS 2015)
# Exercise 2: April 20, 2015

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

March 16, 2015

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 2: Univariate Polynomials

A univariate polynomial $\sum_{i=0}^{n} c_i \cdot x^i$ of degree $n$ can be represented by its $n+1$ coefficients $c_0, \ldots, c_n$. The goal of this exercise is to implement a corresponding class `Polynomial` whose objects represent univariate polynomials with rational coefficients; a coefficient is represented by an object of class `Rational`.

In more detail, the implementation shall be as follows:

1. Implement a class `Rational` with the following interface:

```
class Rational {
public:
  // rational with numerator n and denominator d (default 0/1)
  Rational(int n=0, int d=1);

  // numerator and denominator
  int numerator() const;
  int denominator() const;

  // string representation of this rational
  string str() const = 0;

  // sum and product of this rational and c
  const Rational operator+(const Rational& c) const;
  const Rational operator*(const Rational& c) const;
};
```

   A rational shall be represented in a canonical form where the denominator is positive and numerator and denominator do not have any non-trivial common divisor (use the Euclidean algorithm for reducing). If the constructor is called with $d = 0$, the program aborts.

2. Implement a class `Polynomial` with the following interface:

```
class Polynomial {
public:
  // polynomial with degree+1 coefficients of given numerators/denominators
  // (coefficient n[i]/d[i] belongs to monomial with exponent i)
  Polynomial(int degree, int* n, int* d);

  // constructor with degree+1 coefficients of given rationals
  // (coefficient c[i] belongs to monomial with exponent i)
  Polynomial(int degree, Rational* c);

  // destructor
  ~Polynomial();

  // string representation using x for the variable
  // (don't show zero coefficients unless polynomial is zero)
```

```
  string str(const string& x) const;

  // degree of this polynomial
  int degree() const;

  // evaluate this polynomial on m/n
  const Rational eval(int m, int n) const;

  // sum and product of this polynomial and p
  const Polynomial operator+(const Polynomial& p) const;
  const Polynomial operator*(const Polynomial& p) const;
};
```

The class stores internally an array of the polynomial coefficients as `Rational` objects where the leading coefficient is not zero (the zero polynomial is represented by an array of length zero); the constructors thus ignore trailing zeros in the given arrays. The coefficient array is to be allocated on the heap; the destructor of the class thus frees this array. Please note that the second constructor does not store the array passed as an argument as the result array!

When printing a polynomial, only the monomials with non-zero coefficients are printed (except when the polynomial is zero; in this case "0" is printed).

When adding/multiplying two polynomials, first allocate a temporary `Rational` array for the coefficients of the result polynomial and then fill this array with the appropriate coefficients; finally construct the result polynomial from this array using the second constructor and free the array.

3. The class thus supports the following operations:

```
int numerators[]   = { -5, 2, 0, -3 };
int denumerators[] = {  2, 3, 1, -6 };
Polynomial p(3, numerators, denumerators);
cout << p.str("x");                  // 1/2 x^3 + 2/3 x - 5/2
const Rational r = p.evaluate(1, 2); // evaluate for x = 1/2
cout << r.str();
Polynomial q = p+p; cout << q.str();
Polynomial r = p*q; cout << r.str();
```

Test each classes `Rational` and `Polynomial` in a *comprehensive* way (several calls of each method) including also the calls shown above (print the results and show the program output in the deliverable).