Object-Oriented Programming in C++ (SS 2026)
Exercise 2: April 23, 2026

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

January 19, 2026

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: Polynomials in Distributive Representation

Write a program that implements arithmetic on multivariate polynomials in distributive representation: here a polynomial $p \in K[x_1, \ldots, x_n]$ symbolically written as

$$\sum_{i=1}^{k} c_i x_1^{e_{1,i}} \cdots x_n^{e_{n,i}}$$

is represented by a sequence $[m_1, \ldots, m_k]$ of k monomials where each m_i is a pair

$$\langle c_i, [e_{1,i}, \ldots, e_{n,i}] \rangle$$

of a non-zero coefficient $c_i \in K$ and a sequence $[e_{1,i}, \ldots, e_{n,i}]$ of n exponents (natural numbers) representing the power product of the monomial.

In our program, we take $K := \mathbb{Z}$, i.e., we implement polynomials over the ring of integers; furthermore, we sort the monomials $m_1, \ldots, m_k$ in reverse lexicographic order where monomial m_a occurs before monomial m_b, if for some variable index j we have $e_{a,j} > e_{b,j}$ and for every index $k < j$ we have $e_{a,k} = e_{b,k}$. For instance, the polynomial $3xy^2 + 5x^2y + 7x + 11y + 13$ in $\mathbb{Z}[x, y]$ is represented by the sequence

$$[\langle 5, [2, 1] \rangle, \langle 3, [1, 2] \rangle, \langle 7, [1, 0] \rangle, \langle 11, [0, 1] \rangle, \langle 13, [0, 0] \rangle]$$

Every polynomial has thus a unique representation; please note that the zero polynomial is represented by the empty sequence.

In more detail, write a class `DistPoly` for which it shall be possible to execute these commands:

```
// some exponent vectors ("power products")
int e1[2] = {1,2}; int e2[2] = {2,1}; int e3[2] = {1,0};
int e4[2] = {0,1}; int e5[2] = {0,0}; int e6[2] = {2,2};

// construct zero polynomial in two variables, then add monomials
string vars[2] = {"x","y"};
DistPoly p(2, vars);
p.add(3,e1).add(5,e2).add(7,e3).add(11,e4).add(13,e5);

// construct zero polynomial in two variables, then add monomials
DistPoly q(2, vars);
q.add(11,e4).add(-3,e2).add(2,e6).add(-2,e2);

// print p and q
cout << p.str() << endl;
cout << q.str() << endl;

// set p to p+2*q and print it
DistPoly r = p;
r.add(q).add(q);
p = r;
cout << p.str() << endl;
```

This requires (among others) the definition of the following methods in the class:

```
// zero-polynomial in n variables with given names
DistPoly(int n, string* vars);

// copy constructor, copy assignment operator, destructor
DistPoly(DistPoly &p);
DistPoly& operator=(DistPoly &p);
~DistPoly();

// add new term with given coefficient and exponents to this polynomial
// and return this polynomial
DistPoly& add(int coeff, int* exps);

// add polynomial p to this polynomial and return this polynomial
DistPoly& add(DistPoly &p);

// the string representation of this polynomial
string str();
```

A `DistPoly` object shall be represented by

1. the number and names of the variables,
2. a pointer to a heap-allocated array of monomials in these variables,
3. the length of this array,
4. the number of monomials in this array.

At any time, the array shall hold as many monomials as are indicated by the number value which is less than or equal the length value; these monomials hold coefficients different from 0, are unique with respect to their exponents and are sorted in reverse lexicographic order. It might be a good idea to introduce a class `Mono` such that the monomial array in `DistPoly` holds elements of this type (respectively pointers to such elements, depending on your design choice).

When a new polynomial is created, we allocate an empty array of some default size. When a new monomial is added, we first check its coefficient; if it is zero, the monomial is ignored. Otherwise, we search in the array for the position where

1. either a monomial with the given exponent sequence already occurs; in this case, the given coefficient is added; if the resulting coefficient is zero, the monomial is discarded from the array and all subsequent monomials are shifted to fill the gap;
2. or, if there is no such exponent sequence, a new monomial is to be inserted; the subsequent monomials have to be shifted to make room for the new monomial (if the array is full allocate a fresh array of, e.g., double length).

Please note the following:

- No part of the internal representation of a `DistPoly` object shall be shared with any other object; in particular, the exponent array passed to `add` must be duplicated in the representation.

Furthermore, if a `DistPoly` is copied, a new monomial array (and new exponent arrays) must be created for the new object.

- No memory leaks shall arise from the implementation. As a consequence,
  - the class needs a destructor that frees the memory allocated for the number when the object is destroyed, and
  - the memory allocated for a polynomial must be freed before the polynomial gets assigned a new value.
- Check in the addition of polynomials whether the number and names of variables in both polynomials match; if not abort the program with an error message.
- For printing a polynomial, remember that the interpretation of a polynomial with an empty monomial sequence is 0.

Avoid any code duplication but make extensive use of auxiliary functions (that shall become as far as possible private member functions of `DistPoly`).

Write the declaration of `DistPoly` into a file `DistPoly.h` and the implementation of all non-trivial member functions of `DistPoly` and of into a file `DistPoly.cpp`.

Write a file `DistPolyMain.cpp` that uses `DistPoly` and tests its operations (with uni-, bi-, and three-variate polynomials). Test each operation with at least three test cases that also include special cases (such as adding to a polynomial a zero monomial or a zero polynomial).