Institut für Informatik — TUM
Scientific **C**omputing in **C**omputer **S**cience
R. Wittmann

WT 2015

Tutorial
(Advanced Programming)
Worksheet 12:

Assignment 1: Type generic sorting

In this worksheet we show how we can transform an existing implementation for a certain data type such that it is suitable for a wider class of data types. Please download the stub implementation `sorting.cpp` from moodle. Implement a selection sort algorithm[1] in the provided function `mysort` which takes a reference to a `std::vector` with `double` elements: Here is a short sketch of the algorithm: Start with the first entry of the array.

- Compare the value of the current entry with all remaining entries in the given array.
  - If the current entry is bigger: swap the entries.
  - If the current entry is not bigger: do nothing.
- Go to the next entry in our array.

Repeat this operation for all values in the array. Check your results using the provided test implementation in `main`. The array should now be sorted, with the smallest value at the beginning and the largest value at the very end. If so, then enable the macro `SORT_DONE`.

Next, adapt your implementation such that it works for a wider class of data types using templates. What kind of operation do these data types need to provide so that we can use this sort algorithm? If you have finished the template version enable the macro `ASSIGNMENT3_DONE` in the provided source code.

Homework Assignment 2: Specialization for sorting pointers

Explain what the test code for this assignment does and how it is different to the previous assignment. What information is actually sorted now? Implement a specialized version of the sorting function, which handles arbitrary pointers in a way that we actually get a sorted list of values again.

[1] `http://www.algolist.net/Algorithms/Sorting/Selection_sort`

Class Assignment 3: Class methods with templates

```
// implementation according to worksheet 11 requirements
template <typename DataType> class TArray;
class Base { public: int a; };
class Derived : public Base { public: int b; };
int main() {
    Base b;
    Derived d;
    b = d; // is working

    TArray<Base> base_array(10);
    TArray<Derived> derived_array(10);
    base_array = derived_array; // is not working
    return 0;
}
```

Why is it possible to copy single objects but not arrays of objects?
Propose a solution using additional template parameters.

Assignment 4: Expression templates for polynomials

In this exercise we use expression templates to evaluate polynomials with arbitrary fixed order. Please download `expression_templates.cpp` from moodle which provides a small test setup. Then, work through the following sub tasks which will compute the respective parts of a polynomial of order n:

$$p(x) = \sum_{i=0}^{n} c_i \cdot x^i$$

1. Implement a type generic version for the `add` and `multiply` operations whereas both take two arguments and return the respective result.
2. Implement a type generic version for the `power` (x^n) operation which takes a value as an argument. Use the type generic version of the `multiply` operation for this.
3. Implement a type generic version for computing `PolynomialTerm` ($c \cdot x^n$) which takes a coefficient and the value as an argument. Use the type generic versions of the `multiply` and `power` operations for this.
4. Finally, implement the method `evaluate` in the `Polynomial` object which shall evaluate the polynomial for a given value and a given array of coefficients. You may assume that the respective array is large enough. Only use type generic versions of `add` and `PolynomialTerm`.

Use `const` and `static` modifiers where appropriate. Further, you are not allowed to use loops or explicitly store temporary results.