Solutions to this sheet are due on 14.06.2019 til 14:00. Please hand in a digital version of your answers via e-mail. The e-mail’s subject has to contain cppp19. Do zip-compress your solutions. 

Note: If you copy text elements / code elements from other sources, clearly mark those elements and state the source. Copying solutions from other students is prohibited. All of your files that belong to your solution have to be contained in a single .zip file that is named according to the following naming scheme: <name>_ surname_solution_<X>.zip. Replace <name> and <surname> with your actual name and replace <X> with the number of the exercise sheet. You can look up your results using this link [https://docs.google.com/spreadsheets/d/1Ve7yfS6eZD5XKCI1bxa8W7mpvnlEiaCv61i3P2F4ouO/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1Ve7yfS6eZD5XKCI1bxa8W7mpvnlEiaCv61i3P2F4ouO/edit?usp=sharing)

This exercise sheet will help to familiarize yourself with C/C++’s preprocessor (CPP). You will also learn about programming using templates. Using templates you can write more abstract and generic code that can be used to solve a whole bunch of tasks rather than just one specific one. You can achieve 16 points in total.

Exercise 1.
In this exercise, you have to use C/C++’s preprocessor.

a) Define a preprocessor macro MY_ASSERT(BOOL_EXPR, MESSAGE) that checks if BOOL_EXPR evaluates to true or false. If BOOL_EXPR is evaluated to false print the message MESSAGE as well as the file and the line number this condition failure has been detected to the command line and call exit(-1), which is defined in the <cstdlib> header. The call to exit will exit the program abnormally. (2 P.)

b) Define another preprocessor macro POWERRESULT, BASE, EXPONENT) that computes BASE^EXPONENT (the mathematical power function) and stores the result in RESULT. (2 P.)

c) Why is it not a bright idea to define a macro like #define FAC(N) (N > 1) ? N * FAC(N-1) : 1—using recursion—in order to compute the factorial function? Does this even work? If it goes wrong, why is that? (2 P.)
Exercise 2.
Defining a data type for triples.

a) Define a struct by the name of triple that is capable of storing three variables of arbitrary types!
   Each of the variables may have a different type, say A, B and C.
   (2 P.)

b) Provide the two following member functions for triple:
   - triple(A a, B b, C c); // a simple constructor that initializes the data members
   - friend ostream& operator<< (ostream& os, const triple& t); // an operator to print the data
     members to the command line in a convenient manner
   (2 P.)

Exercise 3.
Consider the bubble sort algorithm from exercise 04.2.a. A possible implementation that you can use for
this exercise is shown here:

```cpp
#include <iostream>
#include <vector>
#include <algorithm> // contains the for_each algorithm and swap
#include <functional> // needed for part b)
using namespace std;

void bubble_sort(vector<int>& v) {
    bool has_swapped;
    size_t n = v.size();
    do {
        has_swapped = false;
        for (size_t i = 0; i < n - 1; ++i) {
            if (v[i] > v[i+1]) {
                swap(v[i], v[i+1]);
                has_swapped = true;
            }
        }
        // After each iteration the biggest element has swapped to the end.
        // Therefore, we can shorten our loop after each iteration.
        −−n;
        // If no swap has taken place, we are done.
    } while (has_swapped);
}

int main() {
    vector<int> v = {10, 9, 8, 7, 6, 5, 4, 1, 3, 2};
    for_each(v.begin(), v.end(), [](int i) { cout << i << " "; }); cout << \n';
    // Do the sorting!
    bubble_sort(v);
    for_each(v.begin(), v.end(), [](int i) { cout << i << " "; }); cout << \n';
    return 0;
}
```

The bubble_sort() implementation in the above is able to sort vectors of integers. Sorting, however, is
a more general task. Given a certain predicate, one can basically sort everything (that can be ordered).
In this task, you will craft a more abstract bubble sort implementation that is able to sort a vector of
"everything".
a) First, change the `bubble_sort()` implementation such that it can operate on everything that implements the `operator<`. In order to do so, make `bubble_sort()` a function template such that it can sort a `vector` of an arbitrary type `T`! Test the function by instantiating a template function that sorts `double` values and check if your function still works correctly.

(3 P.)

b) In order to sort a `vector` of an arbitrary type `T` that does not need to implement `operator<`, adjust the signature of function `bubble_sort()` to receive a second parameter of type `function` that serves as a predicate. The signature looks like: `void bubble_sort(vector<T>& v, function<bool(T, T)> predicate);`

Then, rather than performing a check for `<` in the `if` condition, apply the predicate function to compare two values. An example call of the adjusted `bubble_sort()` is shown in the following:

(3 P.)

```
// ... your adjusted implementation

bool cmp_string_size(const string& a, const string& b) { return a.size() < b.size(); }

int main() {
    vector<string> v = {"A", "BBBB", "CC", "DDDD", "EE", "FFFFFFFF", "G", "HHHH"];
    // sort v according to the predicate 'cmp_string_size'
    bubble_sort<int>(v, cmp_string_size);
    // or you can just pass a cool lambda function and sort in reverse order
    bubble_sort<int>(v, [](const string& a, const string& b) { return a.size() > b.size(); });
    return 0;
}
```

Exercise 4.

This is an optional exercise:

```
template<class T, class... Args>
T add(T t, Args... args) {
    // TODO
}
```

Implement the above function template such that it adds arbitrary many values of arbitrary type. The keyword here is variadic template arguments (use google or a book). Hint: use recursion, you will need a helper function template that represents the trivial case. Having implemented the above and the helper function template, you can use the `add()` function as follows:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    int isum = add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    cout << isum << '\n';
    string s1 = "Hello", s2 = ", ", s3 = "World", s4 = "!";
    string ssun = add(s1, s2, s3, s4);
    cout << ssun << '\n';
    return 0;
}
```

(0 P.)