C++ Programming

Exercise Sheet 02
Software Engineering, EIM-I
Philipp Schubert
October 27, 2017

Solutions to this sheet are due on 03.11.2017 til 14:00. Please hand in a digital version of your answers via e-mail. The e-mails subject has to contain cppp. Do zip-compress your solutions. For questions please send mail or speak to me during the exercises.

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>_solution<XX>.zip. Replace <name> and <surname> with your actual name and replace <XX> with the sheet number the solutions belong to. You can look up your results using this link: https://docs.google.com/spreadsheets/d/1V8rKtimsQS6thKGkTh6CChlv-LwulBIA3RvKAfZSH2M/edit?usp=sharing

During this exercise sheet you will learn how to use functions in C++. Furthermore, you will make yourself familiar with some other container type. At last you will have a quick look into pointers. You can use the code snippets from https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Softwaretechnik/Lehre/CPP_Programming/WS2017_2018/code_02.zip.

You can achieve 16 points in total.

Exercise 1.
In this exercise you will implement some basic linear algebra using C++. In particular, you will implement a few functions that perform some useful operations on mathematical vectors. We will use vector<double> in order to represent a mathematical vector \( v \in \mathbb{R}^n \). Write a program that implements a function for each of the following tasks. Check your function implementations by calling them on small test data like shown in the code snippet below:

```cpp
#include <cmath>
#include <iostream>
#include <vector>
using namespace std;

void print_dvector(const vector<double> &v) {
    for (const double &d : v) {
        cout << d << " ";
    }
    cout << "\n";
}
```
double euclidean_length(const vector<double> &v);
double scalar_product(const vector<double> &v, const vector<double> &w);
vector<double> normalize(vector<double> v);
double euclidean_distance(const vector<double> &v, const vector<double> &w);

int main() {
    vector<double> a = {1, 2, 3};
    vector<double> b = {4, 5, 6};
    // You have to provide the implementations for the four function declarations
    // from above in order to make this code work.
    cout << "length of 'a': " << euclidean_length(a) << '\n';
    cout << "scalar product of 'a' and 'b': " << scalar_product(a, b) << '\n';
    print_dvector(normalize(a));
    cout << "distance between 'a' and 'b': " << euclidean_distance(a, b) << '\n';
    return 0;
}

Exercise 2.
Fibonacci numbers are numbers from an integer sequence, called Fibonacci sequence. Every number in this sequence is the sum of the two preceding ones: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... The Fibonacci sequence $F_n$ can be defined by the following recurrence relation (recursion):

$$F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

Exercise 2.
Fibonacci numbers are numbers from an integer sequence, called Fibonacci sequence. Every number in this sequence is the sum of the two preceding ones: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... The Fibonacci sequence $F_n$ can be defined by the following recurrence relation (recursion):

$$F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

Exercise 2.
Fibonacci numbers are numbers from an integer sequence, called Fibonacci sequence. Every number in this sequence is the sum of the two preceding ones: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... The Fibonacci sequence $F_n$ can be defined by the following recurrence relation (recursion):

$$F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}$$
Exercise 3.
Declare a variable `mymap` of type `std::map<string, int>` which is declared in the standard template library. (Use `#include <map>`) Please refer to [http://en.cppreference.com/w/cpp/container/map](http://en.cppreference.com/w/cpp/container/map) for how to use `std::map`. You can find detailed descriptions as well as example code. You should have a look at the member functions (constructor), `operator[]` and the corresponding examples.

a) Add the following tuples to `mymap` that map a person's name to their age: ("Peter", 40), ("Brian", 4), ("Stewie", 1), ("Chris", 15), ("Meg", 14). (1 P.)

b) Write a function that prints the contents of `mymap` to the command line. (2 P.)

c) Add the tuple ("Lois", 41) to `mymap` and print the contents of the map again. (1 P.)

Exercise 4.
In the lecture we have already learned that pointer and reference types can be quite useful. We also discussed that one can represent points-to relationships as a graph. Consider the following (not very useful) code snippet:

```cpp
int i, j, k;
int *a = &i;
int *b = &k;
int **p = &a;
int **q = &b;
int *c = q;
```

a) Draw the corresponding directed graph, that captures the points-to relations of the above code snippet. Use nodes to represent variables and directed edges to represent points-to information. Write the variables name and its type in each node. (2 P.)

b) Please watch the following video providing an excellent introduction to pointers: [https://youtu.be/Rxvv9krECNw?t=4m18s](https://youtu.be/Rxvv9krECNw?t=4m18s) (0 P.)