

Exercises for the Introductory Course to C++

This document contains all exercises. We encourage to ask questions! Remember:

- There is a folder, containing several files with examples,
- You received the course slides,

thus use these resources! In addition, you may search the internet.

Besides the exercises within this document, there will be a slightly larger programming project to work on towards the end of the course.

Hello World

Exercise 1: Hello World

Task: Implement the “Hello World”-program.

Objective: First glance no structure and compilation.

Solution: 03_hello.cpp

Assigning Input from the Command Line

Exercise 2: Assigning Input

In practice this will hardly be used. However the technique is fundamental to all sorts of input, including files.

Objective: Know how to read from a stream and assign input to a variable.

Task: Write a program, which asks for an integer number. Assign this to a variable (`std::cin` » is the pendant to `std::cout` «). Output the variable. What happens if you accidentally type in a floating point number instead of a whole number?

Solution: 05_input.cpp

Loops and Conditionals I

Exercise 3: Deciding upon Classifiers

In science, we have to do case distinction all the time ... in programming, we have conditionals for that purpose:

Objective: Write a C++ program that classifies a person into an age group based on their age.

Child:	Age < 13
Teenager:	Age >= 13 and < 20
Adult:	Age >= 20 and < 65
Senior:	Age >= 65

Hint: Take the solution from the previous exercise, copy it to a new file, i.e. `classifier.cpp`, and work from there.

Task:

Declare an integer variable `age` and assign it a positive integer value. Use `if/else` statements to classify the person into one of the following groups:

Solution: `06_a_conditionals.cpp`

Exercise 4: Adding numbers I

Task: Add up the first n integer numbers, i.e.

$$1 + 2 + 3 + \dots + n, \quad (1)$$

for $n = 1000$. Compare your result to the formula

$$\frac{n(n+1)}{2} \quad (2)$$

and output if both results are equal or not.

Solution: `06_a_adding.cpp`

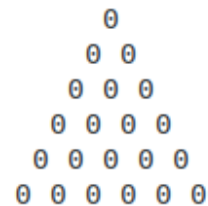
Exercise 5: While Loop

Task: Prompt the user to input a number (use a `float` variable). Check if the input is larger than 10 or not. If it is larger, end the program; if it is smaller ask the user again for an input.

Objective: Apply the `while` loop construct.

Solution: `06_b_while.cpp`

Exercise 6: Pyramide



Task: Using nested loops, produce a figure like this: `0 0 0 0 0 0 0`

Objective: Understand how loops and conditions can be nested to your

advantage.

Hints:

- Start from the top, unlike the Egyptians.
- Use a loop to iterate over all rows.
- And one to fill the spaces to the first '0'.
- And one to enter the '0'.
- For each row, there are less spaces and more '0'.
- For a better look, you might want to output "0 ".

Solution: 06_c_pyramide.cpp

Basic I/O

Exercise 7

[subtitle=Input and Output I] **Task:** Write a program that takes two numbers as inputs and outputs their sum.

Hints:

- Use the input example from exercise 2.
- Remember the general syntax for addition and binding to a variable

```
a = a + b;  
// or  
a += b;  
// or with a new variable  
c = a + b;
```

Solution: 07_a_adding_inputs.cpp

Exercise 8

[subtitle=Adding Numbers II] **Objective:** Combining the afore seen techniques.

Task: Extend the program from exercise 4 to request a number n from the user. Make sure that the user does not provide a negative number nor a number larger than 10000. If the number is invalid, use the command

`return 1;`

to abort your program.

Hints:

- declare a variable for the upper boundary of your loop which is determined by the user. Think about its type.
- remember the if/else syntax:

```
if (condition) {
    // code
} else {
    // alternative code
}
```

- and the for-loop syntax:

```
for (<start>, <condition>, <in- or decrement>) {
    //code
}
```

Note that the <> are a part of the description, not the syntax.

Solution: 07_b_adding_extended.cpp

Exercise 9: Input and Output II

Task: Extend the program from exercise 7 to request to following from the user:

- An integer **op** with values 1, 2, 3 encoding one of the mathematical operations +, −, ·,
- an indefinite number of inputs.

Upon entering 0, the program should display the result. For example, if the user chose **op** as 1 and subsequently entered the numbers 1, 10, 42, the program should compute $1 + 10 + 42$ after the user entered 0 again. Do not forget to check if the value of **op** is within the allowed range!

Solution: 07_c_adding_inputs_extended.cpp

Exercise 10: Adding numbers II

Task: Declare a `std::vector` and ask the user for 5 numbers. Store these numbers as floating numbers in the vector. Output the contents of the vector together with the sum of all numbers.

Solution: 07_d_adding_II.cpp

Functions

Exercise 11: Adding numbers III

Task: Use your code from exercise 10 and write a function that outputs the contents of a vector and returns the sum of the elements of the vector. Use this new function instead of your old way to print the contents directly.

Solution: 08_a_print_sum_vector.cpp

Exercise 12: Quadratic Equation

Task: Given three real numbers a , b , c , the quadratic equation $ax^2 + bx + c = 0$ admits in general two solutions. Recall from school, that the solutions may be computed as

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Recall, that there are three possibilities for the number of solutions:

- If $b^2 - 4ac > 0$ there are two distinct real solutions,
- if $b^2 - 4ac = 0$ there is a unique real solution,
- if $b^2 - 4ac < 0$ there is no real solution.

Write a program, that asks the user for the coefficients a , b , c . Include a function `int num_sol(float a, float b, float c)` that returns the number of real solutions of the quadratic equation. Depending on the number of real solutions, use a `switch` construct to output the number of real solutions and the solutions themselves. Does it make sense to qualify the any part of the function as `inline`, `const` or `constexpr`?

Solution: 08_b_quadratic_equation.cpp

Exercise 13: Quadratic Equation

Task: Write a program, that for 3 different variables different (numerical) types and values calls *one* function `cubic`, which takes a number (n) and returns n^3 . Call this function for all the variables.

Solution: 08_c_template_overloading.cpp