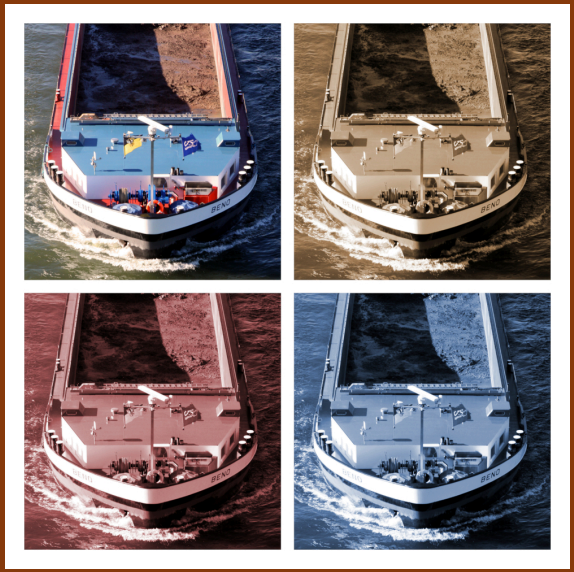
Marc Hensel

# C Programming Workbook

## Exercises, Questions, Lab Assignments, and Exams



HAW HAMBURG

# Foreword

This workbook was originally intended for my students in the Bachelor degree program *Information Engineering* at Hamburg University of Applied Sciences (HAW Hamburg). Its main objective is to recall what we have learned in the lectures and apply it, apply it, apply it. In the end you must build up practical competencies to be prepared for your working life in industry—meaning, you must be able to *do things*, in contrast to merely *knowing*.

What is helpful for my students, may as well be helpful for your students, you as a student, or you being interested in learning the basics of how to write programs in the C programming language. Therefore, please feel free to join us in our effort to learn and improve.

**Exercises and questions** The 86 exercises and 91 questions are aligned with the typical structure of introductory lectures and, equally important, the learning targets of my lecture. Thus, you should work through the theory and this workbook "in parallel". Apart from this, be curious and implement your own software according to your personal interests. The difficulty and scope of the exercises significantly increase in chapters on purpose. Comparatively easy exercises in the first chapters are meant to give you a smooth start in software development with the C programming language. More difficult and complex exercises, in contrast, shall increase your competencies, prepare you for an exam, your further studies, and your professional life.

**Laboratory assignments** The exercises and questions are succeeded by 6 laboratory assignments (i. e., more extensive tasks), some of those building upon another. When working through the exercises you will find hints, when you should be prepared to solve the next lab assignment. These hints shall help you to work through the materials in a good manner, independent from other time constraints such as semester planning. For my students, be very well aware of the following:

- *Preparation:* Although it is not mandatory, I *very strongly* recommend to solve the assignments *on your own* and before the date of the lab. This gives you time to discuss the tasks and solutions in depth and clarify questions with your lab partner. In case you cannot solve the tasks, swap ideas with your team partner and/or other fellow students. During the labs create *one* common solution for your team to present and discuss.
- *Successful participation:* It is mandatory to be present and punctual at *all* labs. Moreover, you are not allowed to take the exam, if you have not passed the labs. Apart from working on the tasks your code *must* comply with the coding style according to Appendix B on page 76. Moreover, you must be able to explain the theory behind the concepts covered in the respective labs.

**Exams** If you want to practice even more, there are 6 exams you may want to give a try.

**Appendix** The appendix of this workbook in meant to help you with practical programming. So far, it contains an overview of the most important rules for software quality (*coding style*) and selected keyboard shortcuts when working with Visual Studio.

**Sample solutions** Sample solutions for all exercises as well as files provided for the sample exams are available on GitHub. Please refer to the appropriate Visual Studio solution and therein to the project stated at the beginning of each exercise.

**Downloads**

`https://github.com/MarcOnTheMoon/coding_learners_c`

Let's begin—enjoy!

Marc Hensel

C Programming Workbook: Exercises, Questions, Lab Assignments, and Exams
Document version: 06.10.2023

# Contents

**Foreword** **3**

**Contents** **5**

## I Exercises and questions 9

**1 Introduction** **10**
1.1 Exercises . . . 10
1.2 Questions . . . 10

**2 Data types** **11**
2.1 Integers and floating point numbers . . . 11
2.2 Keyboard input . . . 12
2.3 Questions . . . 13

**3 Flow control** **14**
3.1 Conditional selections . . . 14
3.2 Loops . . . 14
3.3 Keyboard input . . . 16
3.4 Questions . . . 16

**4 Functions** **17**
4.1 Miscellaneous functions . . . 17
4.2 Random numbers . . . 17
4.3 Mathematical functions . . . 18
4.4 Audio synthesis . . . 19
4.5 Questions . . . 19

**5 Arrays** **21**
5.1 One-dimensional arrays . . . 21
5.2 Multi-dimensional arrays . . . 22
5.3 Functions . . . 23
5.4 Strings . . . 25
5.5 Questions . . . 25

**6 Pointers** **26**
6.1 Fundamentals . . . 26
6.2 Data types . . . 26
6.3 Arrays . . . 27
6.4 Strings . . . 28
6.5 Questions . . . 28

**7 Memory management** **29**
7.1 Mathematical functions . . . 29
7.2 Strings . . . 29
7.3 Arrays . . . 29
7.4 Questions . . . 30

**8 Structures** **31**
8.1 Structures . . . 31
8.2 Enumerations and types . . . 32
8.3 Questions . . . 34

**9 Lists & sorting** **35**
9.1 Exercises . . . 35
9.2 Questions . . . 35

**10 Input & output** **36**
10.1 Exercises . . . 36
10.2 Questions . . . 36

**11 Bit operations** **37**
11.1 Coding and compression (logical operators) . . . 37
11.2 Binary images (logical operators) . . . 38
11.3 Linear filters (bit shifting) . . . 39
11.4 Questions . . . 40

**12 Projects & preprocessor** **41**
12.1 Exercises . . . 41
12.2 Questions . . . 41

**II Laboratory assignments** **42**

**13 Lab 1: Getting started** **43**
13.1 General requirements . . . 43
13.2 Algorithmic operations for integer numbers . . . 43
13.3 Digit sum . . . 44
13.4 Questions . . . 44

**14 Lab 2: Data types & control flow** **45**
14.1 Chessboard fields . . . 45
14.2 Wheat on chessboard problem . . . 45
14.3 Compliance with coding style (quality) . . . 46

**15 Lab 3: Functions** **47**
15.1 Theory: Geographic coordinates and distances . . . 47
15.2 Assignments . . . 49
15.3 Compliance with coding style (quality) . . . 49

**16 Lab 4: Arrays & pointers** **50**
16.1 Route length (1-D arrays) . . . 50
16.2 Maximum distance (2-D arrays) . . . 51
16.3 Compliance with coding style (quality) . . . 51

**17 Lab 5: Memory management & structures** **52**
17.1 Travel planning using *double* arrays . . . 52
17.2 Travel planning using a structure . . . 53
17.3 Compliance with coding style (quality) . . . 53

**18 Lab 6: Structures & file output** **54**
18.1 Check-in . . . 54
18.2 Mandatory requirements . . . 54
18.3 Optional requirements . . . 55
18.4 Compliance with coding style (quality) . . . 56

**III Exams** **57**

**19 Vectors and prime numbers (Winter 2016/17)** **58**
19.1 Expressions . . . 58
19.2 Mathematical 2-D vectors . . . 58
19.3 Prime numbers . . . 59

**20 Complex numbers and Euclidean algorithm (Winter 2017/18)** **61**
20.1 Mixed questions . . . 61
20.2 Complex numbers . . . 61
20.3 Divisors and Euclidean algorithm . . . 62

**21 Array operations (Winter 2018/19)** **64**
21.1 Mixed questions . . . 64
21.2 Array operations . . . 64
21.3 User input and bit operations . . . 65

**22 Euler's number and strings (Summer 2018)** **67**
22.1 Mixed questions . . . 67
22.2 Euler's number . . . 67
22.3 Strings . . . 68

**23 Text analysis and recursive functions (Winter 2019/20)** **69**
23.1 Mixed questions . . . 69
23.2 Text analysis . . . 69
23.3 User input and recursive function . . . 71

**24 Series expansions (Winter 2022/23)** **72**
24.1 Mixed questions . . . 72
24.2 Series expansions . . . 72

**IV Appendix** **74**

**A Visual Studio** **75**

**B Checklist software quality** **76**

**Index** **77**

# Part I

# Exercises and questions

Chapter 1

# Introduction

## 1.1 Exercises

**Exercise 1.** (Project: SolutionFiles)
Get to know the file structure of Visual Studio solutions and projects:

1. Open the sample solution given in the lectures and select *Build / Clean 01a_FirstApplication*.
2. Find the corresponding source code file in the file system of your computer (e. g., using Windows Explorer).
3. Compile the project and find the object file *01a_FirstApplication.obj*.
4. Build the project and find the executable file *01a_FirstApplication.exe*.
5. Select *Build / Clean 01a_FirstApplication* and, again, find the **.obj* and **.exe* files.

**Exercise 2.** (Project: printf)
Print text using *printf()*:

1. Create a project with a source file (extension **.c*).
2. Right-click the project in the solution explorer and select *Set as StartUp Project*.
3. Write the source code such that the program displays your name on the console.
4. Run the program with the symbols `'\n'` appended to your name. Now run the program with `'\n'` appended twice and three times. What is the effect of `'\n'`?
5. Print your name and place of birth in different lines.

**Exercise 3.** (Project: getchar)
Get to know *getchar()*:

1. Copy the source code from *01a_FirstApplication.c* into a new project.
2. Remove the line containing *getchar()* and run the program. What is the effect of that line?
3. Copy and paste the line of code containing *getchar()* so that it appears twice, and after that three times. What is the effect?

## 1.2 Questions

**Question 1.** "C source code is translated into a file that can be executed on every common system (such as Windows or Linux)." Correct or incorrect?

**Question 2.** What does a compiler do?

**Question 3.** What does a linker do?

**Question 4.** Explain the difference between source file, object file, and executable file.

**Question 5.** What is the name of the function that is contained in every C program?

**Question 6.** Where does the execution (initialization excluded) of a C program always start?

**Question 7.** What is the shortest possible statement?

Chapter 2

# Data types

## 2.1 Integers and floating point numbers

**Exercise 4.** (Project: ResistorsSeries)
Write a program that calculates the resistance $R = R_1 + R_2$ of two resistors $R_1 = 150\ \Omega$ and $R_2 = 220\ \Omega$ connected in series and prints the result to the console.

**Exercise 5.** (Project: BinaryWeights)
Write a program that calculates the weights $2^n$ of binary digits for $n \in \mathbb{N}_0$, $n < 8$, and prints them to the console.

**Exercise 6.** (Project: TeamPlayers)
In physical education, pupils of year four shall be partitioned in eight teams for an athletic competition. Write a program to determine

a) the number of pupils in each team and

b) how many pupils remain left over when assuming teams of equal size

in the case that two classes consist of 24 children and the other classes consist of 25 and 26 children, respectively.

**Exercise 7.** (Project: DiodeInt)
An electric current of $I = 20$ mA flows through the protecting series resistor $R = 150\ \Omega$ of an diode. Write a program using the data type *int*, only, which calculates and prints

a) the voltage $U = RI$ at $R$ and

b) the power $P = UI = I^2R$ consumed at the resistor.

**Exercise 8.** (Project: DiodeFloat)
Solve Exercise 7 using the data type *float*.

**Exercise 9.** (Project: ResistorsParallel)
Write a program that calculates the resistance $R$ with

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} = \frac{R_1 + R_2}{R_1 R_2} \tag{2.1}$$

of two resistors $R_1 = 150\ \Omega$ and $R_2 = 220\ \Omega$ connected in parallel and prints the result to the console.

**Exercise 10.** (Project: LeibnizSeries)
According to Gottfried Wilhelm Leibniz the number $\pi$ can be calculated by following series expansion with $N \to \infty$:

$$\pi \approx 4 \cdot \sum_{n=0}^{N} \frac{(-1)^n}{2n+1} = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots + \frac{4 \cdot (-1)^N}{2N+1} \tag{2.2}$$

Write a program that calculates and prints all approximations $N = 0$ to $N = 6$.

(a) Mobile score board including time measurement (prototype)


(b) Android app

Figure 2.1: Sample application using the time format *hh:mm:ss* (Exercise 14)

## 2.2 Keyboard input

**Exercise 11.** (Project: Fahrenheit)
Write a program that prompts users to enter a temperature $T_F$ in Fahrenheit and prints the corresponding temperature $T_C$ in Centigrade according to following formula:

$$T_C = (T_F - 32) \cdot \frac{5}{9} \tag{2.3}$$

**Exercise 12.** (Project: Char2Numeric)
Write a program that prompts users to enter a character and prints the corresponding numeric code to the console.

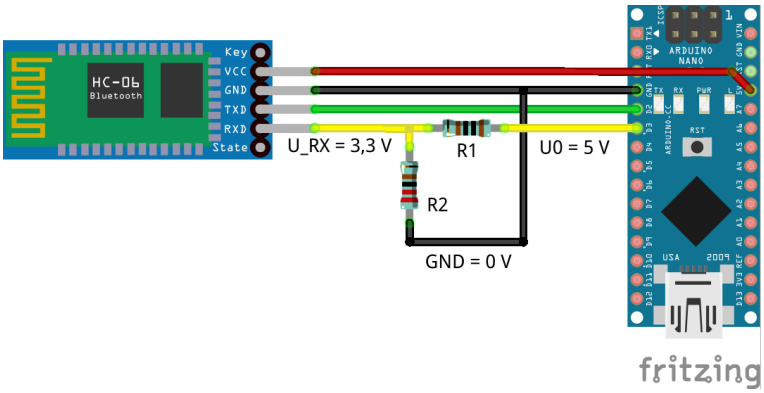
**Exercise 13.** (Project: Lower2Uppercase)
Write a program that prompts users to enter a small letter (a – z) and prints the corresponding capital letter (A – Z) to the console. Try to find a solution that does *not contain a fixed number* to be subtracted from small letters in the source code.

**Exercise 14.** (Project: TimeSeconds)
Write a program that prompts users to enter a time period in seconds. Print the period formatted *hh:mm:ss* (h: *hours*, m: *minutes*, s: *seconds*) to the console. Hint: The format specifier %02d formats an integer value to have at least n digits and adds leading zeros, if required.

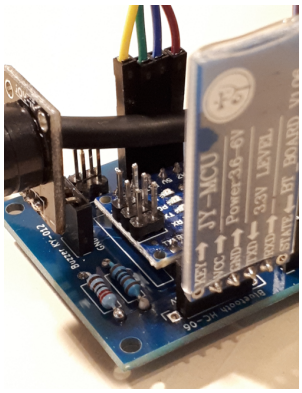
**Exercise 15.** (Project: AverageGrade)
In an school exam, pupils receive grades 1 (excellent) to 6 (poor). Write a program that prompts users to enter the total number of students having received grade 1, 2, 3, ..., 6 and prints the average grade to the console.

**Exercise 16.** (Project: LogicLevel)
When connecting a Bluetooth module HC-06 to an Arduino Nano R3, the voltage level of the RX signal must be reduced from 5 V to about 3.3 V. A simple approach for this is a voltage divider with (Fig. 2.2)

$$U_{RX} = \frac{R_2}{R_1 + R_2} \cdot U_0 \ . \tag{2.4}$$




(a) Bluetooth module HC-06 connected to an Arduino Nano R3



(b) HAW card reader *Tavata*

Figure 2.2: Simple logic level conversion from 5 V to 3.3 V (Exercise 16)

However, taking the current $I_{RX}$ into the *RXD* pin into account, results in

$$U_{RX} = \frac{R_2}{R_1 + R_2} \cdot (U_0 - R_1 I_{RX}) \,. \tag{2.5}$$

Write a program that reads the values of the the resistors $R_1$ and $R_2$ from the keyboard prints $U_{RX}$ for load currents $I_{RX} = 0$ mA and $I_{RX} = 0,5$ mA, respectively. Which of the resistor values 1 kΩ, 1.5 kΩ, 2.2 kΩ, and 4.7 kΩ should you use for $R_1$ and $R_2$, when the logic level $U_{RX}$ must not drop below 3 V?

## 2.3 Questions

**Question 8.** How many different variables can be printed to console using only one call of *printf()*?

**Question 9.** How to print the percentage character `%` to console?

**Question 10.** What is specific about dividing two integers compared to dividing to *float* variables?

**Question 11.** What data type has the result of dividing an *int* by a *double* value?

**Question 12.** What data type has the result of dividing a *double* by an *int* value?

**Question 13.** What is the difference between variable *count* and *&count*?

**Question 14.** What is the difference between the *long int* and *long* data types?

**Question 15.** What is the value of following expression: `(int) 7.1 +3.5`?

**Question 16.** What is the value of following expression: `(3 +4) / 2*3`?

**Question 17.** What is the value of following expression: `3 * 3 +4 / 2`?

**Question 18.** What logical values are represented by the following: 0, 1, 2, 745, -1?

**Question 19.** Name three different ways of adding the value 1 to an integer variable *a*.

**Question 20.** What, do you think, motivated the name *C++*? :-)

**Related laboratory assignment**

Now it is time to work on Lab 1 on page 43.
Make sure that you can solve the lab before you continue.

Chapter 3

# Flow control

## 3.1 Conditional selections

**Exercise 17.** (Project: CheckCharacter)
Write a program that reads a single character from the keyboard and prints to the console, whether the character entered is a digit, small letter, capital letter, or neither a digit nor a letter.

**Exercise 18.** (Project: LeapYear)
The date February 29 exists only in leap years. A year is a leap year, if it can be divided by 4, except for years which can be divided by 100, but not by 400. Write a program that allows users to test whether a specific year is a leap year.

**Exercise 19.** (Project: LeapYearLogical)
Find a solution for Exercise 18 that does not use any *if* statement to determine, whether a given year is a leap year.

## 3.2 Loops

**Exercise 20.** (Project: LoopTypes)
Calculate the sum of the numbers from 1 to 100 using a a) *for*-loop, b) *while*-loop, and c) *do/while*-loop.

**Exercise 21.** (Project: PrintNumbers)
For $n \leq 20$, $n \in \mathbb{N}$, print in ascending order all numbers in the first line, all odd numbers in the second line, and all even numbers in the third line.

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 1  3  5  7  9 11 13 15 17 19
 2  4  6  8 10 12 14 16 18 20
```

**Exercise 22.** (Project: CodeTable)
Print the numeric codes for small and capital letters A – Z in a table. Use the columns *Letter*, *Small*, and *Capital*.

**Exercise 23.** (Project: BinaryWeights)
Implement printing the first eight terms $2^n$ of the binary number system according to Exercise 5 auf Seite 11, however, using a loop.

**Exercise 24.** (Project: RangeShort)
Write a program that determines the numeric range (i. e., the smallest and largest representable number) of the data type *short*. (Hint: Run through the numbers in a *while*-loop. What will happen when the range is exceeded?)

**Exercise 25.** (Project: BankAccount)
On January 1 a bank account has a balance of €1.000,-. How many years will it take to double the balance when the rate of return is 0,35 % per year[1]?

[1] The sad truth is that the balance will not be doubled before 199 years.

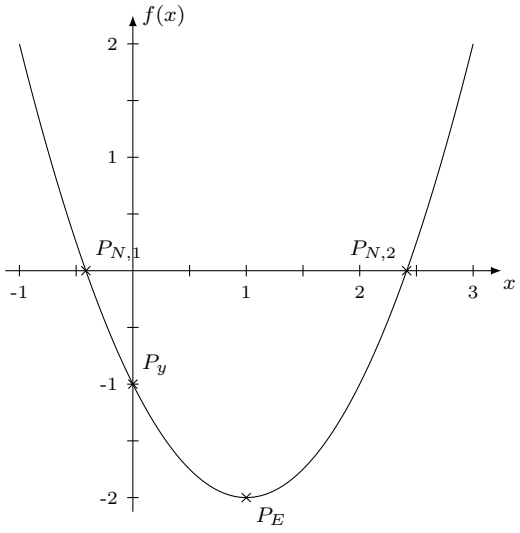



Figure 3.1: $f(x) = x^2 - 2x - 1$ (from M. Hensel: *Kurvendiskussion*, 2. Auflage, 2012)

**Exercise 26.** (Project: LeibnizSeries)
We come back to the series expansion of the number $\pi$ according to Leibniz, see Equation (2.2) on page 11. Print the approximations for $N = 0$ to $N = 10$ and for $N \in \{250, 500, 750, 1000\}$ to the console.

**Exercise 27.** (Project: MathSeries)
Approximate the value of the mathematical series expansion

$$p = \sum_{k=0}^{\infty} 2^{-k} \,. \tag{3.1}$$

After how many summands does $p$ change by less than $\Delta p = 10^{-10}$ per term?

**Exercise 28.** (Project: BisectionMethod)
The mathematical function

$$f(x) = x^2 - 2x - 1 \tag{3.2}$$

has a zero-crossing $x_0$ with $f(x_0) = 0$ on the positive $x$-axis ($P_{N,2}$ in Fig. 3.1). Approximate $x_0$ using the bisection method:

1. Calculate $f(x)$ for $x_l = 0$ and $x_r = 4$. As the signs of $f(x_l)$ and $f(x_r)$ differ, there exists a zero-crossing in the interval $x \in [x_l, x_r]$.

2. Calculate $f(x_m)$ in the center of $x_l$ and $x_r$, i. e., for $x_m = \frac{1}{2}(x_l + x_r)$.

3. Choose either $[x_m, x_r]$ or $[x_l, x_m]$ as next interval of half length, so that the function values $f(x)$ at the interval borders still have different signs.

4. Repeat the procedure from step 2. on, until the interval size is $\Delta x < 10^{-6}$.

## 3.3 Keyboard input

**Exercise 29.** (Project: EchoText)
Write a program where users are asked to enter a complete sentence. When users press the *Enter* key, the program shall print this sentence to the console. (Hint: Call *scanf()* in a loop until there are no more characters in the keyboard buffer.)

**Exercise 30.** (Project: EnterPIN)
Write a program that prompts users to enter an integer PIN. In case the input is equal to the correct PIN, which is stored as constant in the source code, the program confirms that the correct number has been entered. On incorrect input, users are requested to try again. However, the maximum number of attempts is limited to three. Sample output with three incorrect attempts:

```
Please enter pin: 1245
Incorrect. Retry: 2154
Incorrect. Retry: 4512
Whose account are you trying to hack, fellow?!
```

**Exercise 31.** (Project: ReadInteger)
Write a program that prompts users to enter an integer number, reads the input by repeatedly calls of `scanf("%c", &input)`, and stores the number as *int* value. Sample output:

```
Please enter an integer number: 1520
Number scanned    : 1520
Twice that number: 3040
```

## 3.4 Questions

**Question 21.** How is the return value of the *printf()* function interpreted?

**Question 22.** How is the return value of the *scanf()* function interpreted?

**Question 23.** Explain the main difference between the functions *scanf()* and *getchar()*.

**Question 24.** Can a *switch*-statement switch on *char* values? Can it switch on *double* values?

**Question 25.** What is the effect of a *break*-statement within a *switch*-statement?

**Question 26.** What is the effect of a *continue*-statement within a *switch*-statement?

**Question 27.** How often is a body of a *while*-loop, *for*-loop, and *do/while*-loop executed at least? How often is it executed at most?

**Question 28.** When is it best to select a *for*-loop, when to select a *while*-loop?

**Question 29.** A *while*-loop will terminate when an integer variable $k$ exceeds 100. Will the loop ever terminate, if the loop's body contains a *continue*-statement?

**Related laboratory assignment**

Now it is time to work on Lab 2 on page 45.
Make sure that you can solve the lab before you continue.

Chapter 4

# Functions

## 4.1 Miscellaneous functions

**Exercise 32.** (Project: Fahrenheit)
In Exercise 11 on page 12 you were asked to calculate the corresponding temperature in degree Centigrade for a given temperature in Fahrenheit. Write functions *fahrenheit2Centigrade()* and *centigrade2Fahrenheit()* to calculate from one unit into the other and apply both functions in a sample program.

**Exercise 33.** (Project: CheckAlpha)
Write a function *isLetter()* that returns a logical value *true*, if and only if the character passed as argument is a letter.

**Exercise 34.** (Project: ScanBinary)
Write a function *scanBinaryInteger()*, that prompts users to enter a binary pattern consisting of characters '0' and '1' and returns the corresponding integer value. In case of invalid input the function shall return the value -1. Sample output:

```
Enter binary pattern: 10011
Decimal: 19
```

**Exercise 35.** (Project: PrintBinary)
Write a function printBinary() that prints the binary pattern corresponding to a *unsigned* value passed as argument. Sample output:

```
Please enter a non-negative integer: 42
Binary: 0000000000101010
```

(Hint: Start by printing 16 times the binary digit '0' to the console. In a second step, determine and overwrite the actual binary digits moving from right to left. The cursor position can be moved one position the the left by '\b'.)

## 4.2 Random numbers

**Exercise 36.** (Project: Dice)
Implement a function *throwDice()* that returns the result of throwing a dice as random number $n \in \{1, 2, 3, 4, 5, 6\}$. Additionally, write a program that rolls 100 times and prints the resulting numbers in four rows with 25 numbers each, as well as the average number rolled. Following code snipped demonstrates how to generate random numbers:

```
#include <time.h>
#include <stdlib.h>

int main(void)
{
    int randomNumber;

    srand(time(NULL));          // Initialize random number generator
    randomNumber = rand();      // Get next random number

    return 0;
}
```

Sample output:

```
Throws:
6 6 5 5 6 5 1 1 5 3 6 6 2 4 2 6 2 3 4 1 4 1 3 4 5
5 4 3 3 6 6 1 6 1 4 5 6 2 2 1 6 4 3 4 4 3 4 2 6 5
6 3 5 4 4 2 6 4 2 5 5 6 3 1 1 5 5 3 5 5 3 4 3 4 5
3 1 4 4 3 4 6 1 5 1 3 5 3 6 5 1 4 3 2 6 5 3 1 4 6
Average: 3.8
```

## 4.3 Mathematical functions

**Exercise 37.** (Project: LeibnizSeries)
In Exercise 10 on page 11 we have discussed an approximation of $\pi$ by the series expansion according to Leibniz. Write a function *leibnizSeries()* that return the approximated values according to Equation (2.2). The number $N$ of summands shall be passed as argument to the function. Additionally, write a program that prints the resulting values for $N \in \{0, 50, 100, 150, \ldots, 1000\}$.

**Exercise 38.** (Project: Bisection)
In Exercise 28 on page 15 we have approximated a zero-crossing of the mathematical function $f(x) = x^2 - 2x - 1$ by the bisection method. Improve the structure of your source code by introducing following functions:

- *f()* expects a value $x$ as argument and returns the corresponding function value $f(x)$.
- *bisection()* expects the left and right borders $x_l$ and $x_r$ of the starting interval as well as the interval lengths $\Delta x$, at which to stop the method. The function applies the bisection method based on the mathematical values $f(x)$ returned by calls of the C function *f()*. When the interval length is smaller than $\Delta x$ the function returns the center of the remaining interval.

**Exercise 39.** (Project: Newton)
In Exercise 38 we have approximated a zero-crossing of $f(x) = x^2 - 2x - 1$ by the bisection method. Another wide-spread approach is the method by Newton[1]. Starting at a value $x_0$ a zero-crossing is determined iteratively as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4.1}$$

- Implement Newton's method analogous to Exercise 38 in a function *newton()*. It expects the start value $x_0$, the stop condition $\Delta x$, and the maximum number of iterations as arguments. The function ends the iterations and returns the current $x$ value, when the value has changed by less than $\Delta x$ in a step (i. e., $|x_{n+1} - x_n| < \Delta x$) or the maximum number of iterations has been reached.
- Apply the method for $f(x) = x^2 - 2x - 1$ with $f'(x) = 2x - 2$ and the start values $x_0 = 0$ and $x_0 = 4$.

Sample output:

```
Zero-crossing for f(x) = x^2 - 2x - 1 by Newton's method:
Approximated x0    : 2.414213562
Approximated f(x0): 0.000000000
```

[1] `https://de.wikipedia.org/wiki/Newtonverfahren` (Visited on 28.10.2020)

Table 4.1: Frequencies of the C major scale (0-th octave)

| Note | $C_0$ | $D_0$ | $E_0$ | $F_0$ | $G_0$ | $A_0$ | $B_0$ |
|---|---|---|---|---|---|---|---|
| Frequency [Hz] | 16.3516 | 18.3540 | 20.6017 | 21.8268 | 24.4997 | 27.5000 | 30.8677 |

## 4.4 Audio synthesis

**Exercise 40.** (Project: MusicNotes)
Tones of musical instruments consist of sine-shaped waves, with the pitch depending on the wave's frequencies. The faster the oscillation, the "higher" we perceive a tone to be. For the artificial synthesis of music (e. g., by a synthesizer) the corresponding frequencies of musical notes must be known.

In the *C major scale* the notes get alphabetic notations *C*, *D*, *E*, *F*, *G*, *A*, and *B* (in German notation typically *H* instead of *B*) with increasing pitch. To cover the next higher ranges of pitches, the so-called *octave* is appended with increasing digits 0, 1, 2, ...:

$$C_0 \to D_0 \to E_0 \to F_0 \to G_0 \to A_0 \to B_0 \to C_1 \to \cdots \to B_1 \to C_2 \to \cdots \to B_2 \to \cdots$$

Table 4.1 lists the frequencies of the notes in C major scale[2]. The frequency of a note in the next higher octave results by doubling the value, for instance:

$$C_2 = 2 \cdot C_1 = 2 \cdot 2 \cdot C_0 = 4 \cdot 16.3516\,\text{Hz} = 65.4064\,\text{Hz}$$

Write a function *tone2FrequencyHz()* that expects a tone (`'a'` to `'g'` or `'A'` to `'G'`) and the octave (0, 1, 2, ...) as arguments and returns the frequency in Hertz. Use *tone2FrequencyHz()* to print a table for the first to sixth octave to the console. Sample output:

```
Frequencies of musical notes in Hz:

   |      0     |      1     |      2     |      3     |      4     |      5
---+------------+------------+------------+------------+------------+------------
 C |   16.3516  |   32.7032  |   65.4064  |  130.8128  |  261.6256  |  523.2512
 D |   18.3540  |   36.7080  |   73.4160  |  146.8320  |  293.6640  |  587.3280
 E |   20.6017  |   41.2034  |   82.4068  |  164.8136  |  329.6272  |  659.2544
 F |   21.8268  |   43.6536  |   87.3072  |  174.6144  |  349.2288  |  698.4576
 G |   24.4997  |   48.9994  |   97.9988  |  195.9976  |  391.9952  |  783.9904
 A |   27.5000  |   55.0000  |  110.0000  |  220.0000  |  440.0000  |  880.0000
 B |   30.8677  |   61.7354  |  123.4708  |  246.9416  |  493.8832  |  987.7664
```

## 4.5 Questions

**Question 30.** How many functions can exist in a single source code file?

**Question 31.** Explain the difference between an actual argument and a formal parameter?

**Question 32.** Assume you have two functions with the same name, but different parameter lists in a single source code file. Will the code compile successfully?

**Question 33.** What is the impact of parameter names in function prototypes?

**Question 34.** A function has return type *void*. Is it possible to terminate the function before reaching the end of the function body?

**Question 35.** Are the following prototypes equivalent?

[2] `https://de.wikipedia.org/wiki/Frequenzen_der_gleichstufigen_Stimmung` (Visited on 31.10.2020)

```
int func();
int func(void);
```

**Related laboratory assignment**

Now it is time to work on Lab 3 on page 47.
Make sure that you can solve the lab before you continue.

Chapter 5

# Arrays

## 5.1 One-dimensional arrays

**Exercise 41.** (Project: InitZeros)
Write a program that creates and prints two *int* arrays with 100 elements with value 0. Use a loop and a list, respectively, to initialize the values.

**Exercise 42.** (Project: CopyArray)
Write a program that copied the values of an array to a second array of same size. You are free to chose the size and values of the original array.

```
Source: 1 2 5 9 11 20
Copy  : 1 2 5 9 11 20
```

**Exercise 43.** (Project: InvertArray)
Write a program that inverts the order of elements in an array inline (i. e., without using a second array for intermediate results). Sample output:

```
Source  :  1  2  5  9 11 20 26
Inverted: 26 20 11  9  5  2  1
```

**Exercise 44.** (Project: Dice)
In Exercise 36 we have implemented a dice using random numbers. This exercise is meant to check, whether the generated random numbers are roughly equally distributed.

1. Allocate an array with six elements, where the values are equivalent to the overall number of times the result 1, 2, 3, 4, 5, and 6, respectively, have been generated so far.

2. "Roll" the dice, for instance, 100, 10,000, or 1,000,000 times. Store the absolute occurrences of the numbers 1 to 6 in the array.

3. Print the relative frequencies of the numbers 1 to 6 to the console.

Sample output:

```
Number dice throws: 1.0e+06
Relative frequencies:
Result = 1: 16.67 %
Result = 2: 16.69 %
Result = 3: 16.60 %
Result = 4: 16.67 %
Result = 5: 16.71 %
Result = 6: 16.66 %
```

**Exercise 45.** (Project: MusicNotes)
In Exercise 40 we have discussed frequencies of tones, for instance, for the synthesis of notes in music. Modify *tone2FrequencyHz()* such that the function internally uses an array of frequencies in Hertz for the tones $A_0$ to $G_0$.

| Digit | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

Figure 5.1: Control signals of a 7 segment display

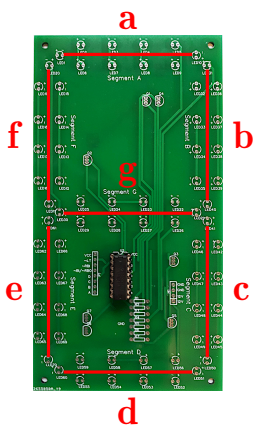



Figure 5.2: PCB of the mobile score board in Fig. 2.1 on page 12

## 5.2 Multi-dimensional arrays

**Exercise 46.** (Project: Clipping)
Colors of single picture elements (*pixel*) in images and videos are often represented by numbers in the range $[0, 255] \subset \mathbb{N}_0$. However, applying image processing can result in values $< 0$ or $> 255$. Write a program that first creates a 2D array that, amongst others, contains values outside the valid range [0, 255]. Run through the array and replace negative values by 0 and values which are too large by 255 (*clipping*). Sample output:

```
Source data:
 -4  -3  -2  -1
  0   1   2   3
252 253 254 255
256 257 258 259

Clipped to [0, 255]:
  0   0   0   0
  0   1   2   3
252 253 254 255
255 255 255 255
```

**Exercise 47.** (Project: SegmentDisplay)
Many technical devices (e. g., the mobile score board in Figure 2.1 on page 12) display digits by 7 segment displays. The segments are typically denoted by letters $a$ to $g$ and switched on or off by dedicated control wires (Fig. 5.1 and 5.2).

Write a function that expects a digit as argument and prints the corresponding values of the control wires $a$ to $g$ to the console. Use the data in Figure 5.1 stored in a 2D matrix. Sample output:

```
Please enter a digit: 6
Control for 7 segment display:

 a | b | c | d | e | f | g
---+---+---+---+---+---+---
 1 | 0 | 1 | 1 | 1 | 1 | 1

Display:
  ---
 |
  ---
```

```
|    |
 ---
```

**Exercise 48.** (Project: MatrixVector)
In mathematics the multiplication of a matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$ and a vector $\mathbf{x} \in \mathbb{R}^3$ are defined as follows:

$$\mathbf{Ax} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1a_{11} + x_2a_{12} + x_3a_{13} \\ x_1a_{21} + x_2a_{22} + x_3a_{23} \\ x_1a_{31} + x_2a_{32} + x_3a_{33} \end{pmatrix} \tag{5.1}$$

Write a program that calculates a matrix-vector multiplication and prints the result to the console. Feel free to choose the coefficients of $\mathbf{A}$ and $\mathbf{x}$. Sample output:

```
          |1 2 0|    |2|    |8|
y = Ax = |0 2 1| * |3| = |7|
          |3 0 1|    |1|    |7|
```

## 5.3 Functions

**Exercise 49.** (Project: MeanValue)
Write a function *mean()* that calculates the average value

$$\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{5.2}$$

of components in a vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$. Sample output:

```
Data: 1.0 2.0 3.0 4.0 5.0 6.5 7.0
Statistical mean = 4.1
```

**Exercise 50.** (Project: StandardDeviation)
Write a function *standardDeviation()* that calculates the standard deviation

$$s = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N} (x_n - \overline{x})^2} \tag{5.3}$$

of components in a vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$. Sample output:

```
Data: 1.0 2.0 3.0 4.0 5.0 6.5 7.0
Statistical standard deviation = 2.2
```

**Exercise 51.** (Project: EqualArrays)
Write a function *isEqualArrays()* that expects two arrays of same size and returns a logical *true*, if and only if the corresponding values of both arrays are equal. Else the function shall return *false*. Sample output:

```
Arrays:
a: 6, 5, 9, 11, 20
b: 6, 5, 9, 11, 20
c: 6, 7, 9, 11, 20

Compare arrays value by value:
Is a same as b?: true
Is a same as c?: false
Is b same as c?: false
```

(a) Image corrupted by noise (b) Filtered image

Figure 5.3: Noise reduction by medial filter (Exercise 53)

**Exercise 52.** (Project: ScalarProduct)
Write a function *scalarProduct()* that calculates and returns the *scalar* or *dot product*

$$\mathbf{x} \cdot \mathbf{y} = \sum_{k=1}^{n} x_k y_k = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \tag{5.4}$$

of two vectors $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$. Sample output:

```
Vectors:
x: (1.0, -2.5, 2.0)
y: (2.5, 1.0, 0.5)

Scalar product xy = 1.0
```

**Exercise 53.** (Project: MedianFilter)
We come back to noise reduction by a binomial filter as presented in the lecture. Apply a so-called *median filter* (Fig. 5.3) to the same data:

1. For each data $a[k]$, sort the values $a[k-1]$, $a[k]$, and $a[k+1]$ ascending.
2. Replace $a[k]$ by the central value in the sorted list.

For instance, median$(3, 7, 1) = 3$ holds, because $1 \leq 3 \leq 7$. Sample output:

```
Original:  95,  91, 211,  97,  89,  96,  94,   3,  91,  94,  92,  96,  93,  97,  94
Median:    95,  95,  97,  97,  96,  94,  94,  91,  91,  92,  94,  93,  96,  94,  94
```

**Exercise 54.** (Project: RotateVector)
In Exercise 48 we have implemented the multiplication of a vector $\mathbf{x} \in \mathbb{R}^3$ with matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$. In the 2D plane $\mathbb{R}^2$ following multiplication rotates the vectors $\mathbf{x}$ by an angle $\alpha$:

$$\mathbf{A}\mathbf{x} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1\cos\alpha - x_2\sin\alpha \\ x_1\sin\alpha + x_2\cos\alpha \end{pmatrix} \tag{5.5}$$

Write a function *printRotatedVector()* that expects a vector $\mathbf{x}$ and an angle $\alpha$ in degrees as arguments and prints the vector rotated by $\alpha$ to the console. Use the library *math.c* for trigonometric functions and be aware that the functions expect angles in radians, not degrees. Sample output:

```
Input vector  : (2.0 3.0)^T
Rotate by     : 90.0 degree
Rotated vector: (-3.0 2.0)^T
```

## 5.4 Strings

**Exercise 55.** (Project: CountLetter)
Write a program that prints the number of occurrences of the letter *a* in a given string to the console. Sample output:

```
"Es war eine Mutter, die hatte 4 Kinder:
den Fruehling, den Sommer, den Herbst und Klaus-Guenter."
The text contains 'a' 3 times.
```

**Exercise 56.** (Project: TextStatistics)
Write a program that prints the number of occurrences of small letters, capital letters, digits, and other characters in a given string to the console. Sample output:

```
"Es war eine Mutter, die hatte 4 Kinder:
den Fruehling, den Sommer, den Herbst und Klaus-Guenter."
Small   letters: 66
Capital letters: 8
Digits         : 1
Other symbols  : 21
```

**Exercise 57.** (Project: ParseKeyValue)
Write a program that prompts users to enter data in the format *key=value* (e. g., `" university =HAW Hamburg"`). Print the parsed inputs for *key* and *value* in separate rows to the console. Sample output:

```
Enter key and value (formatted "key=value"): university=HAW Hamburg
Key   : university
Value : HAW Hamburg
```

## 5.5 Questions

### 5.5.1 Arrays

**Question 36.** What are the first and last indices in every C array?

**Question 37.** How to define an array consisting of five *int* elements and five *float* elements?

**Question 38.** "All elements of an arrays are in connected area of the memory." Correct or incorrect?

**Question 39.** "One can directly assign a list of constant values to an array at any location in the source code." Correct or incorrect?

**Question 40.** What happens if you access element *a[7]* of an array *a* containing seven elements?

**Question 41.** How, if possible, to define an integer array with six dimensions?

**Question 42.** You need to access all elements in a very large 2-dimensional array. Is it faster to run through the matrix row by row or column by column? Explain.

### 5.5.2 Strings

**Question 43.** What is the difference between a array of characters and a string?

**Question 44.** What is difference between `"a"` and `'a'`? What is the respective size in byte?

**Question 45.** What is the size of the *char* array defined by the literal `"Array size"`?

**Question 46.** How many characters will *scanf()* read from the keyboard, if format specifier `%s` is used?

Chapter 6

# Pointers

## 6.1 Fundamentals

**Exercise 58.** (Project: SimplePointer)
Write a program that defines a variable *value* of data type *int* and a pointer *address* referencing *value*.

a) Access the *int* value using *value*. Increment the value stored and print it to the console.

b) Now access the *int* value using *address*. Increment the value stored by adding 1 as well as by using the ++-operator and print it to the console.

Sample output:

```
value   : 0
value++ : 1

Pointer access, only:
Access value : 1
Add value 1  : 2
Increment    : 3
```

**Exercise 59.** (Project: Bisection)
In Exercise 38 on page 18 we have approximated a zero-crossing of the mathematical function $f(x) = x^2 - 2x - 1$ using the bisection method. Modify your solution so that the *bisection()* supplies the calling function with the approximated zero-crossing and the last values of the left and right interval borders $x_l$ and $x_r$. Sample output:

```
Zero-crossing for f(x) = x^2 - 2x - 1 by bisection method:
Stop interval size : 0.000001000
True interval size : 0.000000954
Lower limit  x0    : 2.414213181
Approximated x0    : 2.414213657
Upper limit  x0    : 2.414214134
Approximated f(x0) : 0.000000269
```

## 6.2 Data types

**Exercise 60.** (Project: DataSizes)
Write a program that determines the size of the data types *char* and *int* in standard bytes[1]. Do so by first creating an array with two elements of the respective data type. Determine by how many bytes the memory addresses of the first and second element are apart. Sample output:

```
Address of char[0]: 0093FA80
Address of char[1]: 0093FA81
Size of char in memory: 1 bytes (8 bits each)

Address of int[0]: 0093FA70
Address of int[1]: 0093FA74
Size of int in memory: 4 bytes (8 bits each)
```

[1] That is, each byte consists of eight bits

**Exercise 61.** (Project: DataTypes)
In the following you shall access the individual bytes of an *unsigned* variable by a pointer of type *char**.

a) Declare a variable *value* of type *unsigned* and assign the value 0 to it. Additionally, declare a pointers of data type *unsigned** and *char** and initialize both with the address of *value*. Make sure to fix all compiler warnings.

b) Check that both pointers refer to the identical address in memory.

c) Use the pointer of type *char** to assign the value 1 to the first byte of the *unsigned* variable *value*. Print *value* to the console.

d) Repeat step c) for all other bytes of *value*. Make sure to initialize *value* with 0 before writing the respective byte.

e) Interpret your observations. How are the bytes of the *unsigned* variable stored in memory?

Sample output:

```
Address (unsigned*): 001CFB40
Address (char*)    : 001CFB40

Set byte 1 at address 001CFB40 to 1: 1
Set byte 2 at address 001CFB41 to 1: 256
Set byte 3 at address 001CFB42 to 1: 65536
Set byte 4 at address 001CFB43 to 1: 16777216
```

## 6.3 Arrays

**Exercise 62.** (Project: RotateVector)
In Exercise 54 on page 24 we have discussed a rotation of vectors $\mathbf{x} \in \mathbb{R}^2$ by an angle $\alpha$. Extent your solution by a function *rotateVector()* that performs the rotation. Access the values of the vector to rotate by pointers and pointer operations.

**Exercise 63.** (Project: PrintTransposed)
Write a function that receives a matrix $\mathbf{A} \in \mathbb{Z}^{M \times N}$ and prints the corresponding transposed matrix $\mathbf{A}^T$ to the console. Pass $\mathbf{A}$ as pointer to an one-dimensional array of data type *int* to the function.

**Exercise 64.** (Project: TransposeMatrix)
Write a function *transposeNxN()* that transposes a matrix $\mathbf{A} \in \mathbb{Z}^{N \times N}$. Pass $\mathbf{A}$ as pointer to an one-dimensional array of data type *int*. Sample output:

```
Matrix A:
 11  12  13  14
 21  22  23  24
 31  32  33  34
 41  42  43  44

Matrix A^T:
 11  21  31  41
 12  22  32  42
 13  23  33  43
 14  24  34  44
```

## 6.4 Strings

**Exercise 65.** (Project: StringAnalysis)
Write a function that determines the number of vocals, consonants, and digits in a string passed as argument. Do this by running through the string using a pointer. Sample output:

```
String     : "1, 2 oder 3 - du musst dich entscheiden, drei Felder sind frei!"
Vocals     : 16
Consonants : 28
Digits     : 3
```

**Exercise 66.** (Project: WordCount)
Write a function *numberWords()* that determines the number of words (i. e., character sequences divided by blank spaces) in a string passed as argument. Verify the function using strings entered by users. Sample output:

```
Enter text: I'm falling down, keep falling down like leaves, and it's autumn.
Number of words in text: 11
```

**Exercise 67.** (Project: ParseInteger)
Write a function *toInteger()* that converts a string passed as argument in a value of data type *int*. Sample output:

```
Enter a non-negative integer x: 31
x + 1 = 32
```

## 6.5 Questions

**Question 47.** Pointers store addresses in memory. Explain why pointers are nonetheless associated with data types such as *int* or *double*.

**Question 48.** Given a pointer *ptr* to an *int* variable with value 2, what is the data type and result of the expression `7.1**ptr`?

**Question 49.** Given an array *a[]*, what is the difference between the expressions `*a +2` and `*(a +2)`?

**Question 50.** Given a pointer *ptr* to an array *a[]*, simplify the expression `&*(++ptr -1)`.

**Question 51.** Given an 2-D array *a[][]*, simplify the expression `*(*(a +1) +2)`.

**Question 52.** Given pointers `char *charPtr` and `int *intPtr`, how large is `sizeof(charPtr)` related to `sizeof(intPtr)`?

**Question 53.** Explain the difference between `0`, `'0'`, `"0"`, and `'\0'`.

**Question 54.** What is the effect of passing a string with `'\0'` as 4th, 10th, and the 27th element to *printf()*?

**Question 55.** Given the string in Question 54, what is the effect of passing a *char* pointer to the 7th element to *printf()*?

**Related laboratory assignment**

Now it is time to work on Lab 4 on page 50.
Make sure that you can solve the lab before you continue.

Chapter 7

# Memory management

## 7.1 Mathematical functions

**Exercise 68.** (Project: Newton)
In Exercise 39 on page 18 we have approximated a zero-crossing of $f(x) = x^2 - 2x - 1$ using Newton's method. Modify your solution so that the function *newton()* receives the number $N$ of iterations instead of the stopping condition $\Delta x$, and returns an array containing the approximations $x_1$, $x_2$, ..., $x_N$. Sample output:

```
Zero-crossing for f(x) = x^2 - 2x - 1 by Newton's method:
x_1 = 2.8333333333
x_2 = 2.4621212121
x_3 = 2.4149984299
x_4 = 2.4142137800
x_5 = 2.4142135624
x_6 = 2.4142135624
```

**Exercise 69.** (Project: Bisection)
In Exercise 38 on page 18 we have approximated a zero-crossing of $f(x) = x^2 - 2x - 1$ using the bisection method. Modify your solution so that the function *bisection()* returns a $N \times 2$ matrix, where each row contains the left and right borders $x_l$ and $x_r$ of an iteration step. Sample output:

```
Zero-crossing for f(x) = x^2 - 2x - 1 by bisection method:
Step 0: x0 in [2.0000 , 4.0000]
Step 1: x0 in [2.0000 , 3.0000]
Step 2: x0 in [2.0000 , 2.5000]
Step 3: x0 in [2.2500 , 2.5000]
Step 4: x0 in [2.3750 , 2.5000]
Step 5: x0 in [2.3750 , 2.4375]
Step 6: x0 in [2.4063 , 2.4375]
Step 7: x0 in [2.4063 , 2.4219]
Step 8: x0 in [2.4141 , 2.4219]
Stopped at interval size: 0.010000
```

## 7.2 Strings

**Exercise 70.** (Project: Substring)
Write a function *substring()* that receives a string and indices $m$ and $n$, $m \leq n$ and returns the substring from index $m$ to $n$ included. Sample output:

```
Original string: "The Globe Sessions"
Enter indices of substring (formatted "first:last"): 5:7
Substring: "lob"
```

## 7.3 Arrays

**Exercise 71.** (Project: MergeArrays)
Write a function *mergeArraysSorted()* that receives two arrays, merges them to one, and returns the newly created array. The elements of the arrays passed as arguments to the function are sorted in ascending order in the merged array. Sample output:

```
Array 1: 2, 3, 3, 5, 7, 8
Array 2: 1, 2, 4, 7, 9
Merged : 1, 2, 2, 3, 3, 4, 5, 7, 7, 8, 9
```

**Exercise 72.** (Project: RandomArray)
Write a function *createRandomArray()* that allocates an *int* array and initializes it with random numbers:

- Users are prompted in *main()* to enter the number of elements and the minimal and maximal allowed random values.
- Implement *createRandomArray()* and *main()* in different source files.

Sample output:

```
Enter array size and data range (with spaces: "size min max"): 20 6 16
12, 6, 11, 9, 9, 16, 11, 13, 9, 14, 16, 11, 9, 16, 14, 10, 6, 12, 7, 15
```

**Exercise 73.** (Project: RandomMatrix)
Write a function *createRandomMatrix()* that allocates a 2-D *int* array and initializes it with random numbers:

- Users are prompted in *main()* to enter the number of rows and columns as well as the minimal and maximal allowed random values.
- Implement *createRandomMatrix()* and *main()* in different source files.
- The function *createRandomMatrix()* shall call *malloc()* only once, allocating memory for pointers to the rows as well as the values stored in the matrix. Initialize pointers to the rows appropriately.

Sample output:

```
Enter matrix size and data range (with spaces: "rows columns min max"): 3 4 6 20
 17  12   9  17
 16  20  12   9
 15   8  13  20
```

## 7.4 Questions

**Question 56.** Explain the difference between variable *scope* and *linkage*.

**Question 57.** Can a file scope ("global") variable be used by every function in the file containing its definition?

**Question 58.** Why should a programmer typically use *sizeof* to determine the argument for *malloc()*?

**Question 59.** What is the main difference between the functions *malloc()* and *calloc()*?

**Question 60.** State two different ways to access the second element of an array allocated by:

```
int *ptr = (int *)malloc(4 * sizeof(int));
```

**Question 61.** How could you cause a program to run out of stack memory?

**Question 62.** How could you cause a program to run out of heap memory?

**Question 63.** Is it possible to allocate a triangular 2-D array invoking *malloc()* only once?

Chapter 8

# Structures

## 8.1 Structures

**Exercise 74.** (Project: Vector3D)
Declare a structure representing vectors $\mathbf{x} = (x, y, z)^T \in \mathbb{R}^3$. Furthermore, write a function *scalarProduct()* that returns the scalar or dot product

$$\mathbf{x} \cdot \mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3 \tag{8.1}$$

of two vectors $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^3$. Sample output:

```
v1 = ( 2, -3,  1)^T
v2 = ( 5,  3,  2)^T
v3 = ( 4,  4,  4)^T

Vectors a, b are orthogonal\index{Vector!Orthogonal} for a * b = 0:
<v1, v2> =  3
<v1, v3> =  0
<v2, v3> = 40
```

**Exercise 75.** (Project: ComplexNumbers)
Declare a structure containing the value of a complex number $z = x + jy \in \mathbb{C}$ represented by its real part $x \in \mathbb{R}$ and imaginary part $y \in \mathbb{R}$. Furthermore, write following functions for complex numbers:

- *getComplexAbsolute()* returns the absolute value $|z|$ of $z \in \mathbb{C}$
- *getComplexPhase()* returns the phase (or angle) $\arg(z)$ of $z \in \mathbb{C}$
- *addComplex()* returns the sum $z_1 + z_2$ of complex numbers $z_1$, $z_2 \in \mathbb{C}$

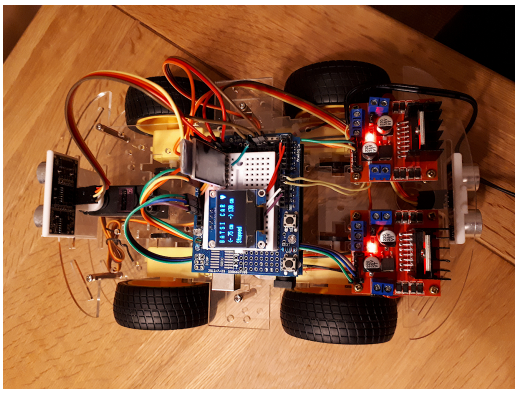
Sample output:

```
z1 =  2.0 -2.0 j
z2 = -0.5 +0.7 j

Absolute: |z1|    = 2.828427
Phase   : arg(z1) = 1.750000 * PI
Sum     : z1 + z2 = 1.5 -1.3 j
```

**Exercise 76.** (Project: Exposure)
Declare a structure including the three exposure parameters *aperture*, *exposure time*, and *ISO sensitivity* of digital photos. Furthermore, write a function *printExposureData()* to print the parameters to the console. The aperture shall be printed as reciprocal value $1/b$ of the f-number $b$. Exposure times $t < 1$ s shall be displayed as fraction $1/T$, $T \in \mathbb{N}$. Sample output:

```
Exposures:
1/2.8  | 1/60  sec | ISO 200
1/5.6  | 1/250 sec | ISO 800
1/16.0 | 3     sec | ISO 100
```



(a) Tilting front (left) and fixed back sensor (right)

(b) Autonomous parking (front, back, and side sensors)

Figure 8.1: Examples for distance sensors in RC cars (Exercise 78 and 79)

## 8.2 Enumerations and types

**Exercise 77.** (Project: SoccerTeam)
In this exercise you are asked to create a simple structure to manage the number of victories, draws, and losses of a soccer team:

- Declare a structure *gameResult* containing symbolic constants WIN, DRAW, and DEFEAT.
- Declare a data type *teamSeason* containing three *int* values representing the number of victories, draws, and losses.
- Write a function *addGameResult()* that receives a team and game result and increases the appropriate counter within the team.
- Write a function *getScore()* that returns the overall points of a team, with three points for each victory, one point per draw, and no points for a lost game.

Sample output when adding a victory, loss, victory, draw, and victory:

```
        win drw def | points
Win    :  1   0   0  |     3
Defeat :  1   0   1  |     3
Win    :  2   0   1  |     6
Draw   :  2   1   1  |     7
Win    :  3   1   1  |    10
```

**Exercise 78.** (Project: ParkSensors)
Ultrasonic distance sensors are a simple and low-cost solution to, for instance, detect obstacles and the distance to objects in robotics and autonomous vehicles (Fig. 8.1). In this exercise you are asked to create a simple structure for sensors mounted to a car:

- Declare a data type *sensorPosition* with integer constants FRONT_LEFT, FRONT_CENTER, FRONT_RIGHT, REAR_LEFT, REAR_CENTER, and REAR_RIGHT. The constants' names correspond to the position of a sensor at a vehicle.
- Declare a data type *distanceSensor* representing a distance sensor. The data type contains the position (type *sensorPosition*) and the last measured distance in cm (type *int*).
- Write a function *getSensorPositionName()* that receives a position of type *sensorPosition* and returns a descriptive string (e. g., "Front left" for FRONT_LEFT). Use a *switch*-statement in your implementation.

- Write a function *printDistances()*, that receives an array of sensors of type *distanceSensor* and prints the corresponding descriptive strings and distances to the console.

Sample output for an array of six sensors:

```
Obstacle detection:
Front left   : 256 cm
Front center: 204 cm
Front right  : 206 cm
Rear left    : 461 cm
Rear center  : 425 cm
Rear right   : 422 cm
```

**Exercise 79.** (Project: ParkPilot)
In Exercise 78 we have implemented a structure representing distance sensors mounted to a vehicle. Building upon this, write a function *displayParkPilot()* that receives an array of sensors of type *distanceSensor* and prints a visualization of the car and the sensors' distances $d$ to the console.

Sample output fot a car with sensors at the front left ($d = 256$ cm), center ($d = 204$ cm), and right ($d = 52$ cm) as well as a sensor at the back center ($d = 425$ cm):

```
Park pilot:

256  204  52
   v-----v
   |     |
   | --- |
   |     |
   |     |
   |     |
   x-----x
     425
```

**Exercise 80.** (Project: Time)
Declare a data type *time* representing the time of the day in hours, minutes, and seconds. Additionally write and apply following functions:

- Function *isTimeValid()* to check, if the data is valid (e. g., minutes in 0 to 59). Declare an enumerated type *timeStatus* with symbolic constants *TIME_OK* and TIME_INVALID for the return value. The value of TIME_INVALID must correspond to the logical value *false*.
- Function *timeFromSec()* accepts the time in seconds elapsed since midnight and returns the corresponding time of type *time*.
- Function *timePeriod()* determines the period between two arguments of data type *time* and returns it as value of type *time*.

Sample output:

```
Valid time   : 1
Invalid times: 0 0 0 0 0 0

Initialize from seconds since midnight:
36318 sec => 10:05:18

Periods:
t0: 09:24:16
t1: 09:25:21
t2: 08:01:00
Period with t0 and t1: 00:01:05
Period with t0 and t2: 01:23:16
```

## 8.3 Questions

**Question 64.** Can members of different structures have the same name?

**Question 65.** A structure *struct pixelRGB* contains three values of type *unsigned char* representing the color components red, green, and blue of an image pixel. How much memory in bytes is allocated at the declaration of the structure?

**Question 66.** Explain the difference between the membership operator (as in `origin.x`) and the indirect membership operator (as in `origin->x`).

**Question 67.** Given a pointer *ptr* to a structure variable. How to access a member variable *x* using the pointer *ptr* and the membership operator (dot '.')?

**Question 68.** Given a structure variable *origin*. How to access a member variable *x* using the variable *origin* and the indirect membership operator?

**Question 69.** Given a structure containing a string member *name* (type *char **), explain the difference between `route[4].name` and `route.name[4]`.

**Question 70.** How to define a completely new data type (e. g., 1-bit integer for Boolean values) using a type definition?

**Related laboratory assignment**

Now it is time to work on Lab 5 on page 52.
Make sure that you can solve the lab before you continue.

Chapter 9

# Lists & sorting

## 9.1 Exercises

Exercises are in preparation and will be added in future.

## 9.2 Questions

### 9.2.1 Linked lists

**Question 71.** How to access the first element of a linked list?

**Question 72.** How to access the last element of a linked list?

**Question 73.** What are advantages of double linked lists compared to linked lists?

**Question 74.** What are disadvantages of double linked lists compared to linked lists?

### 9.2.2 Sorting

**Question 75.** Could bubble sort run from the last to the first element (instead of from first to last)?

**Question 76.** Could quicksort use an element other than the last one as pivot element?

Chapter 10

# Input & output

## 10.1 Exercises

Exercises are in preparation and will be added in future.

## 10.2 Questions

**Question 77.** What console output will *printf()* create, when printing the value 365 using format specifier `%2d`?

**Question 78.** What console output will *printf()* create, when printing the value 25.366 using format specifiers `%02.1f` and `%05.1f`?

**Question 79.** Explain following statement: `putchar(getchar());`

**Question 80.** Given a char variable *c*, explain following statement: `putchar(scanf("%c", &c));`

**Question 81.** What is the difference between the statements `getc(stdin);` and `getchar();`?

**Question 82.** What is the difference between `putc('A', stdout);` and `putchar('A');`?

**Question 83.** You implement a log file functionality for a software component. The file shall contain text information about specific events and be used, for instance, for analysis in case of software malfunctions. What mode should you chose for opening the file?

**Related laboratory assignment**

Now it is time to work on Lab 6 on page 54.
Make sure that you can solve the lab before you continue.

Chapter 11

# Bit operations

## 11.1 Coding and compression (logical operators)

**Exercise 81.** (Project: ParseString)
Write a function *parseStringUnsigned()* that receives a string containing a bit pattern (i. e., characters `'0'` and `'1'`) and returns the corresponding decimal number of data type *unsigned integer*. Sample output:

```
Enter bit pattern (q to quit): 1001
Numeric value: 9
Enter bit pattern (q to quit): 0111
Numeric value: 7
Enter bit pattern (q to quit): 0000100000000
Numeric value: 256
```

**Exercise 82.** (Project: RunLength)
Binary *Run length encoding*[1] (*RLE*) seeks lossless compression of a series of bits. This means, the series (e. g., for storage or transmission) shall be represented by less bits, however, without losing information. To achieve this, RLE utilizes the fact that often there are the same bit values several times in a row. Instead of the individual bits, RLE codes the number of same bit values, alternating for 0 and 1. Let's clarify this by an example:

$$\underbrace{000}_{3}\,\underbrace{1111}_{4}\,\underbrace{0000000}_{7}\,\underbrace{111}_{3}\,\underbrace{0}_{1}\,\underbrace{111111}_{6} \xrightarrow{\text{RLE}} 347316$$

Naturally, the determined numbers must be represented as binary numbers in a computer. When using, for instance, three bits per number, the following 24 bit will be coded by only 18 bit (corresponding to 75 % of the data volume):

$$\underbrace{\underbrace{000}_{3}\,\underbrace{1111}_{4}\,\underbrace{0000000}_{7}\,\underbrace{111}_{3}\,\underbrace{0}_{1}\,\underbrace{111111}_{6}}_{24\text{ Bit}} \xrightarrow{\text{RLE}} 347316 \xrightarrow{3\text{ Bit}} \underbrace{\underbrace{011}_{3}\,\underbrace{100}_{4}\,\underbrace{111}_{7}\,\underbrace{011}_{3}\,\underbrace{001}_{1}\,\underbrace{110}_{6}}_{6\cdot 3=18\text{ Bit}},$$

In this and the next exercise you are asked to implement and apply run length encoding step by step. Let's begin:

a) Create an array of type *unsigned char* containing following six byte:

$$|\underbrace{00001111}_{\text{0x0f}}|\underbrace{00000000}_{\text{0x00}}|\underbrace{11111111}_{\cdots}|\underbrace{00000000}_{\cdots}|\underbrace{00001111}_{\cdots}|\underbrace{11111111}_{\cdots}|$$

b) Write and apply a function *printBitPattern()* to print the bit values of an array of type *unsigned char* to the console.

c) Write and apply a function *runLengthCoding()* that receives an array of type *unsigned char* and prints the corresponding run length code as integer number to the console.

d) Modify *runLengthCoding()* so that the function additionally receives the number of bits used to code the run length (example above: 3 bit) and returns the length of the resulting code in bit.

e) Print the length of the input data and corresponding code in bit to the console.

[1] `https://de.wikipedia.org/wiki/Lauf1%C3%A4ngenkodierung` (Besucht am 21.01.2021)

Sample output for a 4-bit code (i. e., the largest representable code number is $2^4 - 1 = 15$):

```
Original bits: 00001111 00000000 11111111 00000000 00001111 11111111
Code (4 bits): 4*0 | 4*1 | 8*0 | 8*1 | 12*0 | 12*1

Compression:
Input  : 48 bits
Encoded: 24 bits (50.0 %)
```

**Exercise 83.** (Project: RunLengthAdaptive)
In Exercise 82 we have represented run length codes as integer numbers. Modify sour solution as follows:

- The function *runLengthCoding()* receives an additional parameter to control, whether the function shall print to the console or not.
- In case of printing, the function shall output the run length code as bit pattern instead of integer number.
- The *main()* function uses *runLengthCoding()* to determine the number of bits per digit resulting in the shortest run length code. As a second step, it prints the corresponding code as bit pattern to the console.

Sample output:

```
Original bits: 00001111 00000000 11111111 00000000 00001111 11111111

Compression:
Input data: 48 bits
2-bit code: 60 bits (125.0 %)
3-bit code: 42 bits ( 87.5 %)
4-bit code: 24 bits ( 50.0 %)
5-bit code: 30 bits ( 62.5 %)
6-bit code: 36 bits ( 75.0 %)
7-bit code: 42 bits ( 87.5 %)
8-bit code: 48 bits (100.0 %)

Code (4 bits): 0100 0100 1000 1000 1100 1100
```

## 11.2 Binary images (logical operators)

**Exercise 84.** (Project: BinaryImage)
Digital images whose pixels can only have values 0 or 1 (e. g., for “black” and “white”) are called *binary images*. They are used, for instance, to mark areas or *regions* (such like detected objects) in another image. When implementing binary images, in particular memory consumption is of interest. Using one byte per pixel will leave $\frac{7}{8} = 87,5\ \%$ of the memory unused, because only one instead of eight bits are required to represent the pixel values. Thus, in this exercise you are asked to code binary images in such way that each pixel uses only one bit of memory:

- Declare and initialize an array of type *unsigned char*, where each bit represents the value of a pixel.
- Write a function *printBinaryImage()* that receives binary image data as array of type *unsigned char* as well as the number of image rows and columns, and prints the pixel values to the console.

Sample output:

(a) Input image (b) Binary image of detected sign

Figure 11.1: Binary image marking an object (Exercises 84 and 85)

```
Image data (hexadecimal):
0x01 0x23 0x45 0x67
0x89 0xab 0xcd 0xef

Corresponding binary data:
00000001 00100011 01000101 01100111
10001001 10101011 11001101 11101111
```

**Exercise 85.** (Project: BinaryPixels)
In Exercise 84 we have implemented the console output of binary images using only one bit per pixel. In this exercise you are asked to extend your solution as follows:

- Write a function *getBinaryPixel()* that returns the pixel value at the image location $(x, y)$.
- Write a function *setBinaryPixel()* that assigns a value to the pixel at image location $(x, y)$.

Sample output:

```
Binary image data:
00000001 00100011 01000101 01100111
10001001 10101011 11001101 11101111

First byte in row 2 : 10001001
Set 2nd bit to 1    : 11001001
Set 2nd bit to 0    : 10001001
```

## 11.3 Linear filters (bit shifting)

**Exercise 86.** (Project: BinomialFilter1D)
Assume a series $g_1, g_2, \ldots, g_N$ of $N$ numbers to be given. These numbers could, for instance, be the pixel values in a row of a digital image. Application of a linear filter changes the values, i. e., they get replaced by filtered values $\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_N$. In case of a $3 \times 1$ binomial filter the resulting values $\tilde{g}_i$, $1 \leq i \leq N-1$ are the weighted average

$$\tilde{g}_i = \frac{g_{i-1} + 2 \cdot g_i + g_{i+1}}{4} \tag{11.1}$$

at a position with the direct left and right neighboring pixels $g_{i-1}$ and $g_{i+1}$. The first and the last value in the series are not filtered, because these have only one direct neighbor instead of two.


(a) Gray scale image (b) Filtered (c) Difference

Figure 11.2: Example for a 2D binomial filter in image processing (see Exercise 86). (a) Image. (b) Image "blurred" by filter. (c) Absolute difference between (a) and (b). It is evident that filtering has mainly blurred image edges.

- Write a function *binomialFilterPlain()* that receives an array of type *unsigned char* and creates the corresponding filtered values according to Equation (11.1). Do not use bit operations in your implementation.
- Write a corresponding function *binomialFilterBitOps()* that uses fast bit operations where possible.
- Apply both functions to an array initialized with random numbers and verify that both functions lead to identical filtered values.

Figure 11.2 demonstrates a typical application in digital image processing. Note that the weighted averaging of a larger 2-dimension neighborhood has been used.

## 11.4 Questions

**Question 84.** Given integer variables `x = 7` and `y = 10`, what are the results of `x & y` and `x && y`? Explain.

**Question 85.** Do both negation operators `!` and `~` result in the same value when applied to 0? Explain.

**Question 86.** Explain the difference between the OR-operator `|` and the XOR-operator `^`.

**Question 87.** What is the value of variable $y$?

```
unsigned x = 21;
unsigned y = (x &= ~x);
```

**Question 88.** How often does the loop `for (int mask = 1; mask < 8; mask << 1);` run its body?

Chapter 12

# Projects & preprocessor

## 12.1 Exercises

Exercises are in preparation and will be added in future.

## 12.2 Questions

**Question 89.** What is faster: Using a macro with arguments or calling a function with the same functionality?

**Question 90.** A header file contains the variable definition below. How much memory is reserved for this variable in a program during execution?

```
const static char letterA = 'A';
```

**Question 91.** How to skip specific lines of source code, for instance, for debugging a program?

# Part II

# Laboratory assignments

Chapter 13

# Lab 1: Getting started

**Primary learning objectives**

- Create and run sample programs using Visual Studio
- Print data to the screen in a formatted way
- Read integer values from the keyboard
- Apply basic operations to numeric integer data

Welcome—let's get started! I am well-aware that software development might be new to you. Hence, you might experience a sort of "overflow" of new information, related to the C programing language as well as the software tools used to write a C program.

In this lab session, you shall practice creating and running small sample programs using Visual Studio. Don't worry if the programs do not make too much sense, yet. That will change. This lab session is about getting familiar with the most basic steps required for developing C programs.

## 13.1 General requirements

**Project structure** The source code you elaborate for all labs, together, **must be located in a single** Visual Studio solution:

1. Create a Visual Studio solution with your name.
2. For every program written in a lab, create a new project with a descriptive name (e. g., *Lab 1a*, *Lab 1b*, ...) within the solution.

**A comment on comments** For all lab assignments, including future labs, it is *mandatory* that you add appropriate comments to your source code. This includes at least, but is not restricted to:

- A header including a short description of the file contents and author
- Headings to structure the code into logical blocks

**Coding style (quality)** Make sure that all source codes of all labs comply with the requirements on quality in the check list in Appendix B on page 76.

## 13.2 Algorithmic operations for integer numbers

**Literals** Implement a simple calculator for arithmetic operations in file *lab1a.c*:

1. Define two integer variables with values 7 and 3, respectively.
2. Generate the console output below.
   a) The *printf()* function must insert the operand values 7 and 3 from the integer variables.
   b) The results must be calculated using algorithmic operations.

Console output:

```
Simple calculator:
------------------
1st operand: 7
2nd operand: 3

7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7 % 3 = 1
```

**Keyboard input** Implement a modified version of the simple calculator in file *lab1b.c*:

1. Users enter the operands using the keyboard.
2. The console output stays as it is (with the values depending on the user input).

## 13.3 Digit sum

Implement following functionality in file *lab1c.c*:

1. Ask the user to enter his/her year of birth.
2. Calculate and print the digit sum of the entered year.

Example: The digit sum of 2016 is $2 + 0 + 1 + 6 = 9$.

## 13.4 Questions

Answer the following questions in written form:

1. Use the implemented calculator to determine 7 / 10.
   a) What would be the correct result from a mathematical point of view?
   b) What is the result of the calculation? Explain.
2. What happens, if you use 7 and 0 as operands for the calculator? Explain.
3. Digit sum:
   a) Explain the modulo operator % in your own words.
   b) What digit sum results for the 2-digit input 16? Explain.
   c) What digit sum results for the 5-digit input 20169? Explain.

Chapter 14

# Lab 2: Data types & control flow

**Primary learning objectives**

- Select appropriate data types based on expected value range
- Apply statements for flow control
- Print data to the screen in a formatted way

## 14.1 Chessboard fields

Use *for*-loops to print all fields of a chessboard in a 2-dimensional order to the console:

- Rows are labeled 8 (top) to 1 (bottom)
- Columns are labeled $a$ (left) to $h$ (right)

The console output shall look as follows:

```
+----+----+----+----+----+----+----+----+
| a8 | b8 | c8 | d8 | e8 | f8 | g8 | h8 |
+----+----+----+----+----+----+----+----+
| a7 | b7 | c7 | d7 | e7 | f7 | g7 | h7 |
+----+----+----+----+----+----+----+----+
| a6 | b6 | c6 | d6 | e6 | f6 | g6 | h6 |
+----+----+----+----+----+----+----+----+
| a5 | b5 | c5 | d5 | e5 | f5 | g5 | h5 |
+----+----+----+----+----+----+----+----+
| a4 | b4 | c4 | d4 | e4 | f4 | g4 | h4 |
+----+----+----+----+----+----+----+----+
| a3 | b3 | c3 | d3 | e3 | f3 | g3 | h3 |
+----+----+----+----+----+----+----+----+
| a2 | b2 | c2 | d2 | e2 | f2 | g2 | h2 |
+----+----+----+----+----+----+----+----+
| a1 | b1 | c1 | d1 | e1 | f1 | g1 | h1 |
+----+----+----+----+----+----+----+----+
```

## 14.2 Wheat on chessboard problem

There are different variants of the following story. Without making a verdict on which story might be historically true, we will use this one: Sissa ibn Dahir, an Indian minister in the third or fourth century, had invented the game of chess for his ruler. As reward he asked for one grain of wheat being placed on the first field of the chessboard and the number of grains to be doubled with each field (i. e., 1 grain on the first field, 2 on the second field, 4 on the third field, ...). The ruler though this was a modest request – proof him wrong.

**Principal program** Write a program that prints following table:

- Sequential numbering of the chessboard fields (1 to 64)
- Number of grains on the specific field (i. e., 1, 2, 4, 8, ...)
- Sum of grains on all fields up to the specific field (e. g., $1 + 2 + 4 = 7$ for the third field)

The console output shall be like the following:

```
Field | On field |  Sum
------+----------+---------------------
    1 |        1 |        1 (= 1.0e+00)
    2 |        2 |        3 (= 3.0e+00)
    3 |        4 |        7 (= 7.0e+00)
    4 |        8 |       15 (= 1.5e+01)
    5 |       16 |       31 (= 3.1e+01)
    6 |       32 |       63 (= 6.3e+01)
    7 |       64 |      127 (= 1.3e+02)
    8 |      128 |      255 (= 2.6e+02)
------+----------+---------------------
    9 |      256 |      511 (= 5.1e+02)
   10 |      512 |     1023 (= 1.0e+03)
  ... |      ... |      ...
   64 |      ... |      ...
------+----------+---------------------
```

**Data types** Vary the data types used for the number of grains to be *unsigned short*, *unsigned*, *unsigned long* and *unsigned long long*. Explain your observations and determine the number of bits used on your machine to represent each of these data types.

**Overall weight** Determine the overall weight of all wheat grains (assuming a weight of 55 mg / grain). How does this relate to the weight of the worldwide wheat production in 2015, being about 735.8 million tons?

## 14.3 Compliance with coding style (quality)

Make sure that all requirements on software quality according to the check list in Appendix B on page 76 are fulfilled.

Chapter 15

# Lab 3: Functions


**Primary learning objectives**

- Implement, test, and apply functions
- Apply functions of the standard math library

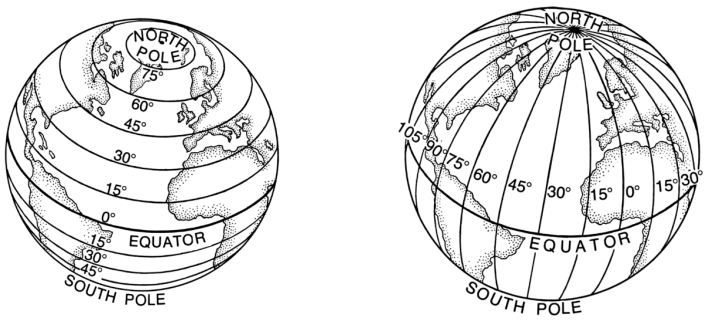
## 15.1 Theory: Geographic coordinates and distances

### 15.1.1 Latitude and longitude

A geographic coordinate $g = (lat, lon)$ is represented by its *latitude* and *longitude* in degree.

- Lines of latitude are parallel to the equator. The range runs from -90° at the South Pole to 90° at the North Pole (Fig. 15.1(a)[1]).
- Lines of longitude run through the North and South Pole and range from -180° to 180° (Fig. 15.1(b)[2]). The so-called prime meridian (longitude 0°) runs through Greenwich, England.

| Name | Orientation | Corresponds to | Range |
|---|---|---|---|
| Latitude *lat* | Parallel to the equator | $y$-coordinate | [-90°, 90°] |
| Longitude *lon* | Through North Pole and South Pole | $x$-coordinate | [-180°, 180°] |




(a) Latitude (b) Longitude

Figure 15.1: Lines of same latitude and longitude

Notes:

- Notice that compared to Euclidean coordinates the $x$- and $y$-coordinates are interchanged.
- Clicking on a location in *Google Maps* will show the corresponding geographic coordinates.

[1] `https://commons.wikimedia.org/wiki/File:Latitude_(PSF).png` (public domain, visited: 18.08.2019)
[2] `https://commons.wikimedia.org/wiki/File:Longitude_(PSF).png` (public domain, visited: 18.08.2019)

### 15.1.2 Local distances

In the following we want to determine the distance between two places on the surface of the Earth. For locations being relatively close to each other we can approximate the distance by neglecting the curvature of the Earth:

1. Determine the distances in km in direction of the latitudes ($\Delta y$) and longitudes ($\Delta x$).
2. Calculate the distance between the places using the theorem of Pythagoras.

Dividing the circumference of the earth into 360° results in 1° corresponding to about 111.3 km. As the distance of neighboring lines of latitude is the same all over the world (Fig. 15.1(a)), a difference of 1° in latitude always corresponds to 111.3 km. The distance between latitudes $lat_1$ and $lat_2$ in km is:

$$\Delta y = 111.3 \cdot |lat_1 - lat_2| \tag{15.1}$$

The distance between neighboring lines of longitude depends on the location on Earth. Directly at the equator (latitude 0°) the distance is 111.3 km. However, neighboring lines of longitude get closer to each other when moving towards the poles (Fig. 15.1(b)). Directly at the North Pole and South Pole all lines of longitude intersect, meaning their distance is 0 km. Following formula models this by a cosine function, which is 1 at the equator (0° latitude) and 0 at the poles (±90° latitude). The cosine's argument is the average latitude of both geographic locations:

$$\Delta x = 111.3 \cdot \cos\left(\frac{lat_1 + lat_2}{2}\right) \cdot |lon_1 - lon_2| \tag{15.2}$$

Overall, the distance $d$ in km between two geographic locations $g_1$ and $g_2$ is (Fig. 15.2):

$$d = \sqrt{\Delta x^2 + \Delta y^2} \tag{15.3}$$

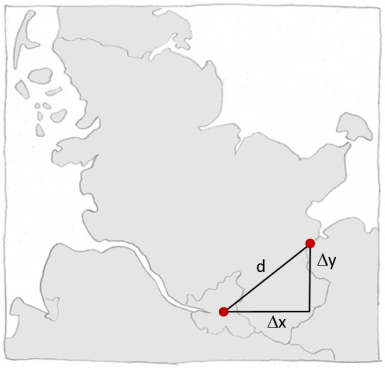



Figure 15.2: Local distance between Hamburg and Lübeck

### 15.1.3 Global distances

A more accurate calculation for the distance between two locations on Earth is given by the following formula[3]. The factor 6378.388 km corresponds to the Earth's radius.

$$d = 6378.388 \cdot \cos^{-1}\left(\sin lat_1 \cdot \sin lat_2 + \cos lat_1 \cdot \cos lat_2 \cdot \cos\left(lon_2 - lon_1\right)\right) \tag{15.4}$$

[3] `https://www.kompf.de/gps/distcalc.html` (visited: 18.08.2019)

## 15.2 Assignments

**Hemispheres**

- Implement a function *isNorthernHemisphere()* that returns a logical *true* only if the geographic location passed to the function is on the northern hemisphere
- Implement a function *isSouthernHemisphere()* accordingly.
- Write a program that uses some test coordinates to verify the correct return values.

**Distances**

- Implement a function *localDistanceKm()* that returns the distance in km between two geographic locations calculated according to (15.3).
- Implement a function *distanceKm()* that returns the distance in km between two geographic locations calculated according to (15.4).
- Verify that the functions return the correct distances between the HAW Hamburg and the Eiffel Tower as given in Table 15.1.
- Determine the missing distances to the HAW Hamburg in Table 15.1.

Notes:

- Include *math.h* for mathematical functions.
- Trigonometrical functions in the math library expect arguments in radians, while geographic coordinates are in degrees (i. e., you must transform latitudes and longitudes to radians).

Table 15.1: Distance of selected locations to HAW Hamburg

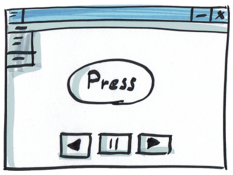
| Location | Latitude | Longitude | Distance (15.4) to HAW [km] | Local distance (15.3) to HAW [km] |
|---|---|---|---|---|
| HAW Hamburg | 53.557078 | 10.023109 | 0.0 | 0.0 |
| Eiffel Tower | 48.858363 | 2.294481 | 750.3 | 750.9 |
| Palma de Mallorca | 39.562553 | 2.661947 | | |
| Las Vegas | 36.156214 | -115.148736 | | |
| Copacabana | -22.971177 | -43.182543 | | |
| Waikiki Beach | 21.281004 | -157.837456 | | |
| Surfer's Paradise | -28.002695 | 153.431781 | | |

## 15.3 Compliance with coding style (quality)

Make sure that all requirements on software quality according to the check list in Appendix B on page 76 are fulfilled.

Chapter 16

# Lab 4: Arrays & pointers




**Primary learning objectives**

- Implement functions working with 1-D and 2-D arrays
- Use pointer types to return more than one value from a function

## 16.1 Route length (1-D arrays)

Let's take a walk—well, at least virtually. Figure 16.1[1] depicts a roundtrip about the HAW Hamburg. The route leads along Lohmühlen park in the direction of lake Alster, around St. Georg hospital, and back to the HAW Hamburg. Determine the overall distance in km.

- Write a function to determine the distance of a route in km.
- Re-use the function *distanceKm()* from lab 3. Do not modify that function.
- Use two arrays, one to hold the latitudes and one to hold the longitudes of the waypoints.
- The provided file *GeoRoute.txt* contains the coordinates of the waypoints.

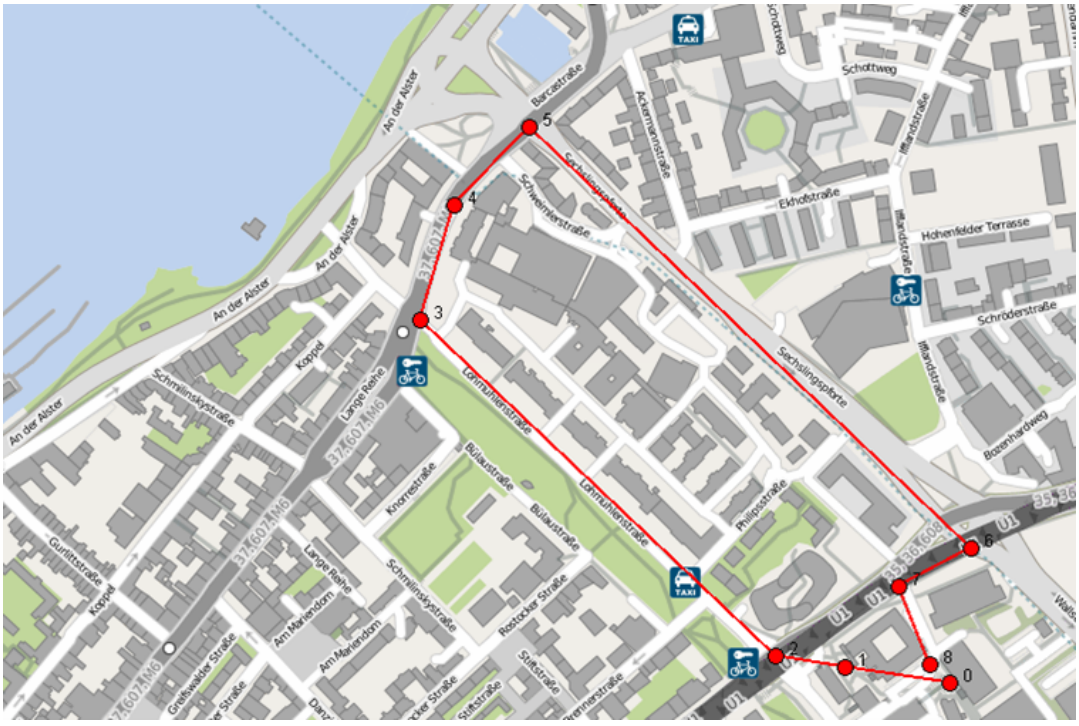



Figure 16.1: Roundtrip at HAW Hamburg

[1] Free OSM map (`http://www.openstreetmap.de/`), created using MOBAC (`http://mobac.sourceforge.net/`)

## 16.2 Maximum distance (2-D arrays)

Table 16.1 contains geographic coordinates of selected locations. Determine which two of these locations are furthest from each other. Proceed as follows:

- Store the locations' names in an array of strings (e. g., `char *names[] = { ... };`).
- Store the geographic coordinates in a $7 \times 2$ matrix. Each row contains the latitude and longitude of a location.
- Write a function that receives, amongst others, a $M \times 2$ matrix of coordinates as parameter. The function shall determine which two locations in the matrix have the largest distance. It returns the maximum distance as well as the row indices of the corresponding locations.
- Print the locations' names and distance to the console.

Sample console output (<...> to be replaced by the results):

```
<Location A> and <Location B> have the largest distance (<Distance> km).
You'd better not walk. Take a flight, instead.
```

Table 16.1: Geographic coordinates of selected locations

| Location | Latitude | Longitude |
|---|---|---|
| HAW Hamburg | 53.557078 | 10.023109 |
| Eiffel Tower | 48.858363 | 2.294481 |
| Palma de Mallorca | 39.562553 | 2.661947 |
| Las Vegas | 36.156214 | -115.148736 |
| Copacabana | -22.971177 | -43.182543 |
| Waikiki Beach | 21.281004 | -157.837456 |
| Surfer's Paradise | -28.002695 | 153.431781 |

## 16.3 Compliance with coding style (quality)

Make sure that all requirements on software quality according to the check list in Appendix B on page 76 are fulfilled.

Chapter 17

# Lab 5: Memory management & structures

**Primary learning objectives**

- Allocate and free dynamic memory as required
- Group data in structures

## 17.1 Travel planning using *double* arrays

Let's book a (virtual) travel! For this, write a program fulfilling the requirements in this and the following section. Table 17.1 contains sample coordinates you may use to very your program. Feel free to choose these or any other coordinates as user input when running your applications.

**Customer requirements:**

- Users shall enter the number of waypoints (i. e., place of departure and the destinations).
- Users shall enter the geographic coordinates of all waypoints.
- After correct user input the program shall print the overall length of the route to the console.

**Technical requirements:**

- In case of invalid input, the application will repeat the request until users enter correct data.
- Dynamically allocate 1-D arrays to hold the latitudes and the longitudes of the waypoints.
- Make sure to free dynamically allocated memory before the application exits.

Sample input/output for incorrect number of waypoints:

```
Enter number of waypoints: three
Try again (expected number >= 0): 3

Enter waypoints as "<latitude> <longitude>":
Waypoint 1: 53.56 10.02
Waypoint 2: 48.86 2.29
Waypoint 3: 39.56 2.66

By taking this route you will travel 1786.2 km.
```

Sample input/output for incorrect geographic coordinate:

```
Enter number of waypoints: 3

Enter waypoints as "<latitude> <longitude>":
Waypoint 1: 53.56 oops
Invalid input (expected "<latitude> <longitude>"): oops
Try again: 53.56 10.02
Waypoint 2: 48.86 2.29
Waypoint 3: 39.56 2.66

By taking this route you will travel 1786.2 km.
```

Table 17.1: Geographic coordinates of selected locations

| Location | Latitude | Longitude |
|---|---|---|
| HAW Hamburg | 53.557078 | 10.023109 |
| Eiffel Tower | 48.858363 | 2.294481 |
| Palma de Mallorca | 39.562553 | 2.661947 |
| Las Vegas | 36.156214 | -115.148736 |
| Copacabana | -22.971177 | -43.182543 |
| Waikiki Beach | 21.281004 | -157.837456 |
| Surfer's Paradise | -28.002695 | 153.431781 |

## 17.2 Travel planning using a structure

**Preparation:**

- Create a copy of your Visual Studio project solving the first assignment.
- Use one of the projects to keep the solution for the first assignment, the other project to create a program implementing the technical requirements below.
- As usual, keep *both* projects in *a single* Visual Studio solution.

**Technical requirements:**

- Introduce a structure *struct geoCoord* holding the latitude and longitude (*not* the name) of a single geographic coordinate.
- Modify your program so that it uses the structure to store the coordinates of the waypoints. The functionality implemented in the first assignment must not be changed.

## 17.3 Compliance with coding style (quality)

Make sure that all requirements on software quality according to the check list in Appendix B on page 76 are fulfilled.

Chapter 18

# Lab 6: Structures & file output

**Primary learning objectives**

- Writing text files (optional)

## 18.1 Check-in

Now that we know about locations and routes, let's check-in for a flight. For this purpose, write a program fulfilling the requirements below. Be aware that *optional* requirements need not be fulfilled to pass the laboratory. Rather, implement these in case you are looking for an additional challenge.

## 18.2 Mandatory requirements

Implement a seating plan for the aircraft model *Bombardier CRJ-200*. The aircraft has 50 seats, overall. Every row, except for the last one, has 4 seats. The last row consists of 2 seats, only. Every seat is either *vacant* or *reserved*.

### 18.2.1 Requirements & hints

Information to be printed to the console:

- Reservation status (*vacant* or *reserved*) for each seat
- Overall number of vacant seats and number of reserved seats

Making reservations:

- Initially all seats are vacant.
- Users are repeatedly prompted to enter a seat to reserve. If the seat is vacant, its status is changed to *reserved* and the console output is updated accordingly.

Hints:

- Use an array of size 50 to represent the aircraft's seats.
- It is permitted to number the seats from 1 to 50. It is not mandatory to denote a seat by its row number and position (e. g., *A* to *D*) within a row.
- Create functions for dedicated tasks (e. g., print seat plan, reserve a seat).
- Do not use global variables.
- You may want to clear the console window before updating the flight plan. Use following function call (declared in *stdlib.h*) on Windows systems:

```
system("cls");
```

### 18.2.2 Sample user input & console output

Initial seating plan with all seats being vacant (indicated by asterisks ∗):

```
Seating plan Bombardier CRJ-200:

  /                \
 /                  \
+                    +
|   1∗   2∗   3∗   4∗ |
|   5∗   6∗   7∗   8∗ |
|   9∗  10∗  11∗  12∗ |
|  13∗  14∗  15∗  16∗ |
|  17∗  18∗  19∗  20∗ |
|  21∗  22∗  23∗  24∗ |
|  25∗  26∗  27∗  28∗ |
|  29∗  30∗  31∗  32∗ |
|  33∗  34∗  35∗  36∗ |
|  37∗  38∗  39∗  40∗ |
|  41∗  42∗  43∗  44∗ |
|  45∗  46∗  47∗  48∗ |
|  49∗  50∗           |
| 0 reserved, 50 vacant∗

Reserve seat(s) (q to quit):
```

Seating plan after the user input “3” and “4” (⇒ reserved seats number 3 and 4):

```
Seating plan Bombardier CRJ-200:

  /                \
 /                  \
+                    +
|   1∗   2∗    3    4  |
|   5∗   6∗   7∗   8∗ |
|   9∗  10∗  11∗  12∗ |
|  13∗  14∗  15∗  16∗ |
|  17∗  18∗  19∗  20∗ |
|  21∗  22∗  23∗  24∗ |
|  25∗  26∗  27∗  28∗ |
|  29∗  30∗  31∗  32∗ |
|  33∗  34∗  35∗  36∗ |
|  37∗  38∗  39∗  40∗ |
|  41∗  42∗  43∗  44∗ |
|  45∗  46∗  47∗  48∗ |
|  49∗  50∗           |
| 2 reserved, 48 vacant∗

Reserve seat(s) (q to quit):
```

## 18.3 Optional requirements

### 18.3.1 Requirements & hints

If you are looking for an additional challenge, implement following adaptations to the mandatory requirements:

- Represent each seat by its row number and position *A* to *D* within the row.
- Allow users to use capital or small letters when reserving a seat (e. g., “7A” or “7a”).
- Skip row number 13 (i. e., row 12 is succeeded by row 14).
- Write the console output to a text file (e. g., *flightPlan.txt*) before exiting the program.

Hints:

- Represent a seat by a structure *seatInfo* and store all 50 seats in an array of type *seatInfo*.
- Let *seatInfo* contain the row number, position in a row (i. e., *A* to *D*), and reservation status.
- Implement a function that writes the seat plan either to the console or to a text file. Use *fprintf()* and pass *stdout* (to write to the console) or the file pointer as argument.

### 18.3.2 Sample user input & console output

Initial seating plan (with all seats being vacant, denoted by asterisks):

```
Seating plan Bombardier CRJ-200:

  /                \
 /                  \
+                    +
|  1A*  1B*  1C*  1D* |
|  2A*  2B*  2C*  2D* |
|  3A*  3B*  3C*  3D* |
|  4A*  4B*  4C*  4D* |
|  5A*  5B*  5C*  5D* |
|  6A*  6B*  6C*  6D* |
|  7A*  7B*  7C*  7D* |
|  8A*  8B*  8C*  8D* |
|  9A*  9B*  9C*  9D* |
| 10A* 10B* 10C* 10D* |
| 11A* 11B* 11C* 11D* |
| 12A* 12B* 12C* 12D* |
| 14A* 14B*           |
| 0 reserved, 50 vacant*

Reserve seat(s) (q to quit):
```

Seating plan after the user input “1C”, “1d”, and “6A”:

```
Seating plan Bombardier CRJ-200:

  /                \
 /                  \
+                    +
|  1A*  1B*  1C   1D  |
|  2A*  2B*  2C*  2D* |
|  3A*  3B*  3C*  3D* |
|  4A*  4B*  4C*  4D* |
|  5A*  5B*  5C*  5D* |
|  6A   6B*  6C*  6D* |
|  7A*  7B*  7C*  7D* |
|  8A*  8B*  8C*  8D* |
|  9A*  9B*  9C*  9D* |
| 10A* 10B* 10C* 10D* |
| 11A* 11B* 11C* 11D* |
| 12A* 12B* 12C* 12D* |
| 14A* 14B*           |
| 3 reserved, 47 vacant*

Reserve seat(s) (q to quit):
```

## 18.4 Compliance with coding style (quality)

Make sure that all requirements on software quality according to the check list in Appendix B on page 76 are fulfilled.

# Part III

# Exams

Chapter 19

# Vectors and prime numbers (Winter 2016/17)

Available time: 180 minutes

## 19.1 Expressions

Following definitions are given:

```
int x = 3;
int a[] = { 4, 3, 2, 1 };
```

For each row in the table below, state the data type and value of the evaluated expression.

| *Expression* | *Data type of result* | *Evaluates to value* |
|---|---|---|
| `10 / 4` | | |
| `10 / 4.` | | |
| `++x / 2` | | |
| `(int)1.6 +2.4 - (3 < 1)` | | |
| `sizeof("1")` | | |
| `*a +3` | | |
| `4 +(4 << 1)` | | |
| `(1 << 2)\| 3` | | |
| `~0 == !0` | | |

## 19.2 Mathematical 2-D vectors

- Implement your solution in the file *question2.c* within the provided project Question 2.
- Do not modify the *main()* function or provided function prototypes.

### 19.2.1 Assignment

Implement functionality for mathematical vectors $\mathbf{a} = (x, y)^T$ with components $x$ and $y$. Vectors shall be stored in arrays of type double and size 2. Your implementation must fulfill all requirements stated in the following.

1. Define the constant DIM, which is used in *main()*, with an appropriate value.
2. Implement *printVector2D()* such that *main()* generates the console output below.
3. Implement the function *vectorLength()* returning the length $|\mathbf{a}| = \sqrt{x^2 + y^2}$ of a vector.
4. Implement the function *scaleVector2D()* that scales (i. e., multiplies) a vector a with a floating-point number $k$ of type *double*.

5. Implement the function *dotProduct2D()* that calculates and returns the dot product $\mathbf{a}_1 \cdot \mathbf{a}_2 = x_1x_2 + y_1y_2$ of two vectors $\mathbf{a}_1 = (x_1, y_1)^T$ and $\mathbf{a}_2 = (x_2, y_2)^T$.

6. Add missing function prototypes for all functions except for *main()*.

7. Add all required include directives.

8. The parameter declarations of all functions except for *scaleVector2D()* must ensure that a function cannot change the values of arrays that are passed to the function.

9. The *main()* function must generate console output exactly as stated below.

### 19.2.2 Expected console output

```
Vectors:
a1 = (3.0, 4.0)
a2 = (-2.0, 6.0)

Vector length:
|a1| = 5.0

Scale a2 with 0.5:
a2 = (-1.0, 3.0)

Dot product:
<a1, a2> = 9.0
```

## 19.3 Prime numbers

- Implement your solution in the file *question3.c* within the provided project Question 3.
- You must not modify the provided function prototypes.

### 19.3.1 Overview

Write a program that prints all prime numbers in the range from 1 to $N$ to the console. Your implementation must fulfill all requirements stated in the following sections.

Mathematical background: An integer number $k \geq 2$ is a prime number, if it has no positive divisor other than 1 and itself. For instance, the number range from 1 to 10 contains the prime numbers 2, 3, 5, and 7. The integer 7 is a prime number, because it can be divided by 1 and 7, only. In contrast, 9 is not a prime number, because it can be divided by 3 (in addition to 1 and 9).

### 19.3.2 Type definition and even numbers

1. Define an enumerated type named *boolean*. The enumeration defines constants FALSE with value 0 and TRUE with value 1.

2. Implement the function *isEven()*, which returns a value corresponding to the logical true, only if the argument can be divided by 2. (Note that numbers 2, 4, 6, 8, ... are called even numbers, while numbers 1, 3, 5, 7, ... are called odd numbers.)

### 19.3.3 Prime numbers

3. Implement the function *isPrimeNumber()*, which returns the constant TRUE, only if the integer argument $k$ is a prime number. Implement the function as follows:
   (a) Return FALSE, if the number is too small ($k \leq 1$).
   (b) Return FALSE, if the number is even and $\geq 4$. Use the function *isEven()*.
   (c) Run through all odd numbers $m$ with $3 \leq m < \frac{k}{2}$, skipping even numbers. Return FALSE, if $k$ can be divided by $m$ without remainder.
   (d) Else return TRUE.

### 19.3.4 Main function

Implement the following functionality in the *main()* function. The sample user input and console output below shall clarify the requirements.

4. Users are asked to enter a positive integer number.
5. While the user input is not valid, users are asked to retry.
6. The keyboard line buffer is emptied after each user input.
7. The program prints to the console all prime numbers in the range from 1 to the number entered by the user, included.
8. The prime numbers are printed in rows of 10 numbers each.
9. The numbers are aligned to the right in each column.
10. The number of prime numbers within the range is printed to the console.

### 19.3.5 Sample user input and console output

Invalid user input:

```
Enter maximum number to test: 0
Invalid input, must be positive integer. Retry: HAW Hamburg
Invalid input, must be positive integer. Retry: 11
Prime numbers in [1, 11]:
   2    3    5    7   11
There are 5 prime numbers in [1, 11].
```

Rows and alignment:

```
Enter maximum number to test: 250
Prime numbers in [1, 250]:
   2    3    5    7   11   13   17   19   23   29
  31   37   41   43   47   53   59   61   67   71
  73   79   83   89   97  101  103  107  109  113
 127  131  137  139  149  151  157  163  167  173
 179  181  191  193  197  199  211  223  227  229
 233  239  241
There are 53 prime numbers in [1, 250].
```

Chapter 20

# Complex numbers and Euclidean algorithm

**(Winter 2017/18)**

Available time: 180 minutes

## 20.1 Mixed questions

For each of the statements below, indicate by a cross (X) in the appropriate column, whether the statement is true or not true. There is no further explanation required. (Note carefully that incorrect answers will reduce the achieved points.)

| *Statement* | *True* | *Not true* |
|---|---|---|
| The ANSI C standard allows to declare variables wherever they are required. | | |
| The size (in bits) of *int* variables may be different for different platforms. | | |
| The *scanf()* function returns the number of items successfully read from the keyboard input. | | |
| C interprets the integer value -1 as logical value *false*. | | |
| A *switch* block runs as infinite loop, if it does not contain a *break* statement. | | |
| The body of a *do/while* loop is executed at least once. | | |
| The function prototypes `int functionName();` and `int functionName(void);` are equivalent. | | |
| For a function `int functionName(int a)` ... the parameter variable *a* is defined only within the function's body. | | |
| The identifier ("'name"') of an array is a pointer to the first element of the array. | | |
| Both, the string `"a"` and the character `'a'`, require 1 byte of memory. | | |
| A pointer of type *double* (e. g., `double *ptr;`) requires more memory than a pointer of type *short* (e. g., `short *ptr;`). | | |
| File scope ("'global"') variables are known in the entire file, even if they are defined at the end of the file. | | |
| The function *calloc()* sets all bits of the allocated memory to the value 0. | | |
| Structures can contain members of different data types (e. g., int and float). | | |
| The logical operators && and & have the same effect. | | |

## 20.2 Complex numbers

- Implement your solution in the file *question2.c* within the provided project Question 2.
- Do not modify the *main()* function or provided function prototypes.

### 20.2.1 Assignment

Implement functionality for mathematical complex numbers $z = x + jy$ with real part $x$ and imaginary part $y$. Your implementation must fulfill all requirements stated in the following.

1. Define a data type complex, which is a structure of two *double* variables named *real* and *imag*.
2. Implement the function *absolute()* that returns the absolute value $|z| = \sqrt{x^2 + y^2}$ of a complex number $z$.
3. Implement the function *add()* that returns the sum $a + b$ of two complex numbers $a$ and $b$.
4. Implement the function *multiply()* that returns the product $a \cdot b$ of two complex numbers $a$ and $b$.
5. Add all required include directives.
6. The *main()* function must generate console output exactly as stated below.

### 20.2.2 Expected console output

```
Input data:
z1 = 3.0 +4.0j
z2 = 2.0 -1.0j

Absolute values:
|z1| = 5.000
|z2| = 2.236

Mathematical operations:
z1 + z2 = 5.0 +3.0j
z1 * z2 = 10.0 +5.0j
```

## 20.3 Divisors and Euclidean algorithm

- Implement your solution in the file *question3.c* within the provided project Question 3.
- You must not modify the provided function prototypes.

### 20.3.1 Overview

A natural number $k$ is a divisor of a natural number $m$, if $k$ divides $m$ without remainder. For instance, the number $m = 18$ can be divided by 1, 2, 3, 6, and 9. The greatest common divisor (*gcd*) of two natural numbers $m$ and $n$ is the largest number that divides both, $m$ and $n$, without remainder. For instance, 18 and 12 can both be divided by 6. As there is no larger number that divides 18 as well as 12, their greatest common divisor is $\gcd(18, 12) = 6$.

Implement functions to determine divisors and greatest common divisors according to the requirements in the following subsections. (Remark: The described method for the greatest common divisor is based on the Euclidean algorithm and is not optimal.)

### 20.3.2 Greatest common divisor

1. Implement a function *sortDescending()* that sorts two integer values in descending order. The function receives pointers to two integer variables $a$ and $b$. If required, it swaps the values of the variables so that $a \geq b$ holds.
2. Implement a function *greatestCommonDivisor()*, which returns the gcd of two integer variables $m$ and $n$. Implement the function as follows:

(a) Use the function *sortDescending()* so that $m$ contains the larger of the two numbers $m$ and $n$ (i. e., $m \geq n$).

(b) Calculate the remainder $r$ of the division $\frac{m}{n}$.

(c) Return the smaller number $n$, if the remainder $r$ is 0.

(d) Else repeat steps 1 to 3 with the values of the smaller number $n$ and the remainder $r$.

### 20.3.3 Divisors

3. Implement a function *getNumberOfDivisors()* that returns the number of divisors (e. g., 5 for $m = 18$) of a natural number $m$.

4. Implement a function *newArrayOfDivisors()* that allocates and returns an integer array containing all divisors (e. g., 1, 2, 3, 6, 9 for $m = 18$) of a natural number $m$. The allocated memory must have the correct size. The divisors must be sorted in ascending order.

### 20.3.4 Main function

Implement the following functionality in the *main()* function:

5. Users are asked to enter two positive integer numbers.

6. While the user input is not valid, users are asked to retry.

7. The keyboard line buffer is emptied after each user input.

8. The program allocates an array containing the divisors of the first number entered by the user. For this, use the appropriate function from the previous sections.

9. The program prints the divisors of the first number entered by the user to the console.

10. The program prints the greatest common divisor of the numbers entered by the user to the console.

11. Ensure that no memory leaks exist when the program terminates.

For clarification of the requirements, find sample output (with sample user input in **bold font**) below:

```
Enter two positive integer numbers (comma-separated): 12 eighteen
Invalid input. Retry: 12 and 18
Invalid input. Retry: 12,18
Divisors of 12: 1 2 3 4 6
Greatest common devisor: gcd(12, 18) = 6
```

Chapter 21

# Array operations (Winter 2018/19)

Available time: 180 minutes

## 21.1 Mixed questions

For each of the statements below, indicate by a cross (X) in the appropriate column, whether the statement is true or not true. There is no further explanation required. (Note carefully that incorrect answers will reduce the achieved points.)

| *Statement* | *True* | *Not true* |
|---|---|---|
| A source file includes *stdio.h* and uses *printf()*. The object file created from this source file contains the machine code of the *printf()* function. | | |
| The *scanf()* function need not distinguish between *float* and *double*, because both data types are floating-point numbers. | | |
| Every *for*-loop can be written as a functionally equivalent *while*-loop. | | |
| Identifiers (”‘names”’) of function parameters need not be the same for a function’s (i) prototype and (ii) definition (i. e., implementation). | | |
| The size of an array cannot be increased after its definition. However, it can be decreased as a smaller array requires less memory. | | |
| Let *ptr* be a pointer variable of a 2-byte data type. If *ptr* contains the value 0x0086E64A, the value of the expression ++ptr is 0x0086E64B. | | |
| A program will crash while executing *malloc()*, if there is not enough memory available. | | |
| The first element in an enumeration can have any int value, including negative values. | | |
| Let $x$ and $y$ be *int* variables. The function call scanf("%d,%d", &x, &y) returns the value 1 (indicating success) for the keyboard input 3,4abcd. | | |
| For variables $x$ of data type *unsigned int*, the expression (x >> 1)<< 1 == x is always *true*. | | |

## 21.2 Array operations

Implement your solution in the file *question2.c* within the provided project Question 2.

### 21.2.1 Assignment

Implement functions according to the following requirements:

1. Implement *printArray()* that prints the elements of an *int* array to the console.
2. The formatting of *printArray()* must match the formatting in Section 21.2.3 below (i. e., print values as comma-separated list in brackets [ ]).
3. Implement *cloneArray()* that creates and returns a copy of an existing *int* array, which is passed to the function as argument.
4. If memory allocation fails, *cloneArray()* ends the program with EXIT_FAILURE.
5. Implement *swap()* to swap (exchange) the values of two *int* variables passed to the function as parameters.

6. Implement *invertArray()* to reverse the order of values stored in an *int* array. Do not allocate a new array. Example: An array containing `{1, 2, 3, 4}` will contain `{4, 3, 2, 1}` after the function call.
7. The function *invertArray()* shall use *swap()* to exchange values.
8. Where appropriate, ensure that values in arrays passed to a function as argument cannot be modified by the function.

### 21.2.2 Funktion *main()*

Implement the *main()* function to fulfill following requirements:

9. Declare an *int* array of size 9 and initialize its values in ascending order 1, 2, ..., 9.
10. Define a variable which stores the number of elements in the array. Do not initialize it with the constant 9, but determine the size by using an appropriate operator.
11. Print the array to the console using *printArray()*.
12. Create a clone using *cloneArray()* and print it to the console.
13. For the cloned array, invert the order of the elements using *invertArray()* and print the array to the console.
14. Ensure that no memory leaks exist.
15. Use appropriate comments to structure your source code (including logical sections within *main()* and *cloneArray()*).

### 21.2.3 Expected console output

For clarification of the requirements, find the expected console output of *main()* below:

```
Input   : [1, 2, 3, 4, 5, 6, 7, 8, 9]
Clone   : [1, 2, 3, 4, 5, 6, 7, 8, 9]
Inverted: [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

## 21.3 User input and bit operations

Implement your solution in the file *question3.c* within the provided project Question 3.

### 21.3.1 Assignment

Implement an application to calculate and print values $2^n$ with $n$ being non-negative integer numbers in the range $0, 1, 2, \ldots, N$. The program must fulfill the following requirements:

1. Define an *int* constant containing the maximum allowed exponent $N$ (e. g., 16).
2. Ask users to enter $n$ in the range from 0 to the maximum allowed exponent $N$.
3. Use the *scanf()* function to read $n$ from the keyboard.
4. While the user input is an invalid integer value, users are asked to retry.
5. If the user input is not an integer, the program terminates.
6. The keyboard line buffer is emptied after every user input.
7. If user input is valid, print the value $2^n$ to the console.
8. Calculate the value $2^n$ by bit operation(s). Do not use a function call, loop, or such.

### 21.3.2 Expected console output

For clarification of the requirements, find sample output (with sample user input in **bold font**) below:

```
Calculation of 2^n with n in [0,16]:
Enter n: 17
Incorrect input: Not in [0,16]
Enter n: 0
2^0 = 1
Enter n: 8
2^8 = 256
Enter n: q
Press any key to quit.
```

Chapter 22

# Euler's number and strings (Summer 2018)

Available time: 180 minutes

## 22.1 Mixed questions

For each of the statements below, indicate by a cross (X) in the appropriate column, whether the statement is true or not true. There is no further explanation required. (Note carefully that incorrect answers will reduce the achieved points.)

| *Statement* | *True* | *Not true* |
|---|---|---|
| Only the executable files differ for different platforms (e. g., Windows and Linux), but the object files are the same. | | |
| Integer division rounds the result to the nearest integer value. | | |
| The integer value -1 represents a logical *true*. | | |
| The expressions `'0'` and `"0"` require the same space in memory. | | |
| The prototypes `int func();` and `int func(void);` are equivalent. | | |
| The size of a pointer variable is the same for all data types. | | |
| Given an array, one can increase its size by adding elements. | | |
| Members in different structures may have the same identifier. | | |

## 22.2 Euler's number

Implement your solution in the file *question2.c* within the provided project Question 2.

### 22.2.1 Euler's number

Euler's number $e$ can be approximated by the series

$$e = \sum_{k=0}^{n} \frac{1}{k!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

with

$$m! = \prod_{k=1}^{m} k = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot m$$

for $m \leq 1$ and $0! = 1$.

1. Implement the function *eulerSeries()* using the provided prototype. The function returns the approximation of Euler's number using above formula. The function parameter corresponds to $n$ in above formula.

### 22.2.2 Main function

Implement the *main()* function to fulfill following requirements:

2. Users are asked to enter the parameter $n$ to use for the approximation.
3. Use the *scanf()* function to read input from the keyboard.
4. While the user input is not valid, users are asked to retry.
5. The keyboard line buffer is emptied after each user input.
6. The program prints the approximated numbers for $n$ in 0 to the value entered by the user. The output must be formatted exactly as the output in the section below.

### 22.2.3 Expected console output

For clarification of the requirements, find sample output (with sample user input in **bold font**) below:

```
Maximum n to use: not a number
Input must be a non-negative integer. Retry: 4
Approx. of Euler's number:
n = 0: 1.000000
n = 1: 2.000000
n = 2: 2.500000
n = 3: 2.666667
n = 4: 2.708333
```

## 22.3 Strings

Implement your solution in the file *question3.c* within the provided project Question 3.

### 22.3.1 Assignment

1. Implement the function *invertString()* using the provided prototype. The function returns a new string, which contains the string passed as argument to the function in reverse order.

   Example: `invertString("abcde")` returns a string containing `"edcba"`.

2. Implement the function *countChar()* using the provided prototype. The function returns the number of occurrences of the character argument in the string argument.

   Example: `countChar("abaac", 'a')` returns the value three.

3. Implement the *main()* function to create console output exactly as stated below. The function must use *invertString()* and *countChar()* to generate the respective results for printing.
4. Exit the program in case memory allocation failed.
5. Make sure the program does not contain memory leaks.
6. Make sure that functions cannot modify their string arguments.

### 22.3.2 Expected console output

For clarification of the requirements, find the expected console output of *main()* below:

```
Original : Welcome to IE-SO1!
Inverted : !1OS-EI ot emocleW
Count 'e': 2
```

Chapter 23

# Text analysis and recursive functions (Winter 2019/20)

Available time: 180 minutes

## 23.1 Mixed questions

For each of the statements below, indicate by a cross (X) in the appropriate column, whether the statement is true or not true. There is no further explanation required. (Note carefully that incorrect answers will reduce the achieved points.)

| *Statement* | *True* | *Not true* |
|---|---|---|
| C is portable between computer platforms. Hence, an executable file build under Windows will also run on a Linux system. | | |
| According to the C standard, all the integer types *short*, *int*, and *long* could have the same size. | | |
| Every *switch* block must contain a *default* label. | | |
| A function call to a function that does not declare a variable (parameter or local variable) does not require any memory. | | |
| C supports 5-dimensional arrays. | | |
| The effect of *const* differs for variables of type *int* and *int**. | | |
| File scope variables can only be used within the file where they are defined. They cannot be accessed from other files. | | |
| Type definitions allow to define new data types such as a 7-bit integer. | | |
| The value of an unsigned integer $x$ could be changed by the expression `(x << 1)>> 1`. | | |
| Preprocessor directives such as `#define` are processed by the compiler. | | |

## 23.2 Text analysis

Some texts are easy to read (e. g. children's books), others are difficult to understand (e. g., scientific papers). A related numeric indicator is the so-called *Flesch Reading Ease Score* (*FRES*)[1]:

$$\text{FRES} = 206.835 - 1.015 \cdot \frac{\text{Number of words}}{\text{Number of sentences}} - 84.6 \cdot \frac{\text{Number of syllables}}{\text{Number of words}}$$

Implement an approximation of the FRES by fulfilling the following requirements. Implement your solution in the file *question2.c* within the provided project Question 2.

### 23.2.1 Data type

1. Declare a structure containing four *int* variables and a *double* variable. The identifiers must match the usage in the provided function *printTextProperties()*.

2. Define a data type called *properties* as alias for the structure declared in above requirement.

[1] `https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests` (Access: 29.12.2019)

### 23.2.2 Statistical analysis

3. Implement *getCharacterCount()* returning the number of occurrences of a specific character in a string. (Example: `getCharacterCount("challenge", 'e')` returns 2, because the string contains the character 'e' two times.)

4. Implement *getNumberOfWords()* returning an estimation of the number of words in a string. Assume that words are separated by a blank space (' ') and use *getCharacterCount()*.

5. Implement *getNumberOfSentences()* returning an estimation of the number of sentences in a string. Assume that the number of sentences corresponds to the number of the characters period ('.'), semicolon (';'), exclamation mark ('!'), and question mark ('?') and use *getCharacterCount()*.

6. Implement *getNumberOfSyllablesInWord()* returning an estimation of the number of syllables in a word. Use the following heuristics:

   (a) Words containing $\leq 3$ characters have 1 syllable.

   (b) For words containing $> 3$ characters the number of syllables corresponds to the number of vowels (a, e, i, o, u) in the word.

7. Implement *getNumberOfSyllables()* returning an estimation of the number of syllables in a string. The implementation shall use the function *strtok()* declared in *string.h* and work as follows:

   (a) Create a copy of the string. Do not allocate more memory than required.

   (b) Get a pointer to the first word in copy by calling `strtok(copy, " ")`.

   (c) Get pointers to next words by repeatedly calling `strtok(NULL, " ")`. The function returns *NULL* when there is no further word.

   (d) Call *getNumberOfSyllablesInWord()* for each word to determine and sum up the number of syllables.

8. Implement *getTextProperties()* to return a structure of type properties containing the statistical values of the string text passed to the function. Where possible, use the functions from the requirements above to determine the values. Calculate readability according to the FRES (see formula above).

### 23.2.3 Quality

9. Exit the application with the value `EXIT_FAILURE` in case a memory allocation fails.

10. Ensure that no memory leaks exist.

11. Where appropriate, ensure that values in arrays passed to a function as argument cannot be modified by the function.

### 23.2.4 Expected console output

For clarification of the requirements as well as for test purposes, find the expected console output of the provided *main()* function below:

```
Test string "No one said; it would. be easy! AaEe IiOoUu?":
Characters  : 44
Words       : 9
Sentences   : 4
Syllables   : 20
Readability : 16.6 out of 100
```

```
William Lyon Phelps, The Pleasure of Books:
Characters  : 402
Words       : 72
Sentences   : 6
Syllables   : 123
Readability : 50.1 out of 100
```

## 23.3 User input and recursive function

Implement your solution in the file *question3.c* within the provided project Question 3.

### 23.3.1 Functions

Implement the functions *clearKeyboardBuffer()* and *arraySum()* to fulfill the following requirements:

1. The function *clearKeyboardBuffer()* removes all characters from the keyboard buffer.
2. The function *arraySum()* returns the sum of all elements contained in an *int* array.
3. The function *arraySum()* uses recursive function calls to calculate the sum.

### 23.3.2 Function *main()*

Implement the *main()* function to fulfill following requirements:

4. Ask users to enter 3 comma-separated integer numbers.
5. Use a single call (instead of three calls) of *scanf()* to read all three comma-separated *int* values from the keyboard into an array of size 3.
6. While the user input is invalid, users are asked to retry.
7. The keyboard buffer is emptied after every user input.
8. Calculate the sum of all elements in the array and print the result to the console.

### 23.3.3 Expected console output

For clarification of the requirements, find sample output (with sample user input in **bold font**) below:

```
Enter three integers formatted x,y,z: 3,1,oops
Invalid input. Retry: 3 1 2
Invalid input. Retry: 3,1,2
Sum = 6
```

Chapter 24

# Series expansions (Winter 2022/23)

Available time: 90 minutes

## 24.1 Mixed questions

For each of the statements below, indicate by a cross (X) in the appropriate column, whether the statement is true or not true. There is no further explanation required. (Note carefully that incorrect answers will reduce the achieved points.)

| *Statement* | *True* | *Not true* |
|---|---|---|
| Depending on the platform, the data types *int* and *long* may have the same size. | | |
| C interprets the integer value -1 as logical value *true*. | | |
| The *scanf()* function needs not distinguish *int* and *long*, because both data types are integer numbers. | | |
| A *switch* block runs as infinite loop if it does not contain a *break* statement. | | |
| The terms `"a"` and `'a'` are equivalent. | | |
| The pointer variable defined by `const int* a = NULL;` can be assigned new values at any time, although it is declared as *const*. | | |
| Pointers of type *long* (e.g., `long *ptr;`) require the same memory as pointers of type *short* (e.g., `short *ptr;`). | | |
| Given integer variables $a$ and $b$, the expressions `a && b` and `a & b` are equivalent. | | |

## 24.2 Series expansions

### 24.2.1 Series expansion and principal value

The mathematical function $\sin\alpha$ can be approximated by the series expansion

$$\sin\alpha = \alpha \cdot \prod_{k=1}^{K} \left(1 - \frac{\alpha^2}{k^2 \cdot \pi^2}\right)$$

with $K \to \infty$ and the product operator

$$\prod_{i=1}^{N} a_i = a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_N.$$

The sine function is periodic in $2\pi$, $\sin\alpha = \sin(\alpha + n \cdot 2\pi)$, with natural numbers $n \in \mathbb{Z}$. The value $\alpha$ with $0 \leq \alpha < 2\pi$ is called the *principal value*.

1. Create a new Visual Studio Solution with your last name (i.e., file <*name*>*.sln*). Within the solution, create new project and C source file.
2. The function *sine()* approximates and returns $\sin\alpha$ according to above formula.
3. The function *principalValue()* returns the principal value (i. e., the equivalent value in the range with $0 \leq \alpha < 2\pi$) of its argument.
4. The appropriate constant in *math.h* is used for all occurrences of $\pi$.

### 24.2.2 Application

Implement a console application according to following requirements:

5. Users are asked to enter the values $x$ and $K_{max} > 0$ for the series expansion of $\sin(x \cdot \pi)$ according to above formula. You must use *scanf()*, not *scanf_s()*.
6. As long as the user input is not valid, users are asked to retry.
7. For each valid user input, print the approximations for $K = 1, 2, \ldots, K_{max}$ calculated by the function *sine()* to the console.
8. For each valid user input, print the principal value of $\alpha = x \cdot \pi$ to the console. Use *principalValue()* to determine the value.
9. After each valid user input, users are asked to quit the application or enter new values. The application quits, only if the letters q or Q are entered.
10. Requirement 10. The keyboard buffer is emptied after every user input.

### 24.2.3 Expected console output

For clarification of the requirements, find sample output (with sample user input in **bold font**) below:

```
Series expansion for sin(x * PI) with k > 0 terms.
Please enter <x>,<k>: 3, -1
Invalid input. Retry: 2.5, 10
Approximations of sin(2.50 * pi):
  k =  1: sin(x * PI) = -41.233404
  k =  2: sin(x * PI) = 23.193790
  k =  3: sin(x * PI) = 7.086991
  k =  4: sin(x * PI) = 4.318635
  k =  5: sin(x * PI) = 3.238976
  k =  6: sin(x * PI) = 2.676654
  k =  7: sin(x * PI) = 2.335244
  k =  8: sin(x * PI) = 2.107193
  k =  9: sin(x * PI) = 1.944601
  k = 10: sin(x * PI) = 1.823063
Principal value: 0.50 * pi
Enter <q> to quit, any other key to continue.

Series expansion for sin(x * PI) with k > 0 terms.
Please enter <x>,<k>: 0.75, 5
Approximations of sin(0.75 * pi):
  k =  1: sin(x * PI) = 1.030835
  k =  2: sin(x * PI) = 0.885874
  k =  3: sin(x * PI) = 0.830507
  k =  4: sin(x * PI) = 0.801309
  k =  5: sin(x * PI) = 0.783280
Principal value: 0.75 * pi
Enter <q> to quit, any other key to continue.
```

# Part IV

# Appendix

Appendix A

# Visual Studio

Table A.1 lists selected useful keyboard shortcuts when working with Visual Studio. Note that comma-separated shortcuts shall be typed in quick succession.

Table A.1: Selected keyboard shortcuts

| Shortcut | Functionality |
|---|---|
| *Ctrl-F5* | Compile, link, and execute without debugger |
| *F5* | Compile, link, and execute with debugger |
| *Ctrl-R*, *Ctrl-R* | Rename (e. g., variable or function) |
| *Ctrl-K*, *Ctrl-D* | Format file (e. g., correct indentation) |
| *Ctrl-K*, *Ctrl-C* | Comment selection (`/* ... */`) |
| *Ctrl-K*, *Ctrl-U* | Uncomment selection |

**Appendix B**

# Checklist software quality

| *Aspect* | *Ok* |
|---|---|
| The code compiles *without* warnings. | ☐ |
| Identifiers are in English language. | ☐ |
| Identifiers are meaningful. | ☐ |
| Identifiers for variables and functions begin with a small letter.<br>Words are separated using *CamelCase* (**not** underscores). | ☐ |
| Identifiers for constants consist of capital letters.<br>Words are separated using underscores. | ☐ |
| Opening and closing braces are in a new line.<br>Opening and closing braces are followed by a new line. | ☐ |
| There is a comment as headline before logical sections inside a function. | ☐ |
| Blocks and such are indented by a tabulator per level. | ☐ |
| There is an empty line between following sections:<br>preprocessor directives, global variables, prototypes, and functions. | ☐ |
| Functions are separated by an empty line. | ☐ |
| Logical blocks are separated by an empty line. | ☐ |
| There is a blank (“space”) between operands and operators. | ☐ |
| No more than *one* empty line in a row.<br>No more than *one* blank (“space”) in a row. | ☐ |
| Structure of source code files:<br>1. Include directives<br>2. Prototypes<br>3. Global variables<br>4. Main function<br>5. Other functions | ☐ |

# Index


$\pi$
    Leibnitz series, 11, 15, 18
7 segment display, *see* Display

Aperture, 31
Arduino, 12
Argument
    Actual, 19
Array
    Copy, 21
    Dimensions, 25
    Equal, 23
    Invert order, 21
    Memory, 25
    Merge sorted, 29
    Multi-dimensional, 22
    One-dimensional, 21
    Random numbers, 30
Audio
    Synthesis, 19
    Tone, 19, 21
Autonomous parking, 32
Autonomous vehicles, 32
Average value, 23

Bank account, 14
Binary numbers, 11, 14
Binary pattern, 17
Binary to integer, 17
Binomial filter, *see* Filter
Bisection method, *see* Zero-crossing
Bit operation, 37, 65
Bluetooth, 12
Bubble sort, *see* Sorting

C++, 13
Calculator, 43
Card reader, 13
Centigrade, *see* Temperature
Character
    Capital letter, 12, 14
    Digit, 14
    Letter, 17
    Numeric code, 12, 14
    Small letter, 12, 14
Chessboard, 45
    Wheat, 45
Clipping, 22
Coding, 37
    Run length encoding, 37, 38
Coding style, 43, 76
Comments, 43
Compiler, 10
Complex number, 31, 61
    Absolute, 31, 62
    Imaginary part, 31
    Phase, 31
    Product, 62
    Real part, 31
    Sum, 62
Compression, 37
    Lossless, 37
Current, *see* Electric current

Data type
    boolean, 59
    char, 16, 25, 26, 28
    char *, 34
    char*, 27
    Definition, 34
    distanceSensor, 32, 33
    double, 13, 16, 28, 52, 58, 61, 62, 64, 69
    float, 11, 13, 25, 64
    gameResult, 32
    int, 11, 13, 21, 25–28, 30, 32, 61, 64, 65, 69, 71, 72
    int*, 69
    long, 13, 69, 72
    long int, 13
    properties, 69
    seatInfo, 56
    sensorPosition, 32
    short, 14, 61, 69, 72
    struct geoCoord, 53
    struct pixelRGB, 34
    teamSeason, 32
    time, 33
    TIME_OK, 33
    timeStatus, 33
    unsigned, 17, 27, 46
    unsigned char, 34, 37, 38, 40
    unsigned int, 64
    unsigned integer, 37
    unsigned long, 46
    unsigned long long, 46
    unsigned short, 46
    unsigned*, 27
    void, 19
Data types, 11
Debugging, 41
Dice, 17, 21
Diode, 11
Display
    7 segment, 22
Distance sensor, *see* Sensor
Divisors, 61
Dot product, *see* Vector

Earth
    Circumference, 48
    Radius, 48
Electric
    Current, 13
Electric current, 11
Electric power, 11
Euclidean algorithm, 61
Euler's number, 67
Exposure parameters, 31
Exposure time, 31

f-number, 31
Fahrenheit, *see* Temperature
File
    *.c, 10
    *.exe, 10
    *.obj, 10
    <name>.sln, 72
    01a_FirstApplication.c, 10
    01a_FirstApplication.exe, 10
    01a_FirstApplication.obj, 10
    Executable, 10
    flightPlan.txt, 55
    GeoRoute.txt, 50
    Header, 41
    lab1a.c, 43
    lab1b.c, 44
    lab1c.c, 44
    Log, 36
    math.c, 24
    math.h, 49, 72
    Object, 10
    question2.c, 58, 61, 64, 67, 69
    question3.c, 59, 62, 65, 68, 71
    Source, 10
    stdio.h, 64
    stdlib.h, 54
    string.h, 70
    Text, 54
Filter
    Binomial, 24, 39
    Median, 24
Flesch Reading Ease Score, 69
Frequency, 19, 21
FRES, *see* Flesch Reading Ease Score
Function
    absolute(), 62
    add(), 62
    addComplex(), 31
    addGameResult(), 32
    arraySum(), 71
    binomialFilterBitOps(), 40
    binomialFilterPlain(), 40
    bisection(), 18, 26, 29
    calloc(), 30, 61
    centigrade2Fahrenheit(), 17
    clearKeyboardBuffer(), 71
    cloneArray(), 64, 65
    countChar(), 68
    createRandomArray(), 30
    createRandomMatrix(), 30
    displayParkPilot(), 33
    distanceKm(), 49, 50
    dotProduct2D(), 59
    eulerSeries(), 67
    f(), 18
    fahrenheit2Centigrade(), 17
    fprintf(), 56
    getBinaryPixel(), 39
    getchar(), 10, 16
    getCharacterCount(), 70
    getComplexAbsolute(), 31
    getComplexPhase(), 31
    getNumberOfDivisors(), 63
    getNumberOfSentences(), 70
    getNumberOfSyllables(), 70
    getNumberOfSyllablesInWord(), 70
    getNumberOfWords(), 70
    getScore(), 32
    getSensorPositionName(), 32
    getTextProperties(), 70
    greatestCommonDivisor(), 62
    invertArray(), 65
    invertString(), 68
    isEqualArrays(), 23
    isEven(), 59, 60
    isLetter(), 17
    isNorthernHemisphere(), 49
    isPrimeNumber(), 60
    isSouthernHemisphere(), 49
    isTimeValid(), 33
    leibnizSeries(), 18
    localDistanceKm(), 49
    main(), 30, 38, 58–63, 65, 68, 70, 71
    malloc(), 30, 64
    mean(), 23
    mergeArraysSorted(), 29
    multiply(), 62
    newArrayOfDivisors(), 63
    newton(), 18, 29
    numberWords(), 28
    parseStringUnsigned(), 37
    principalValue(), 72, 73
    printArray(), 64, 65
    printBinaryImage(), 38
    printBitPattern(), 37
    printDistances(), 33

printExposureData(), 31
printf(), 10, 13, 16, 28, 36, 43, 64
printRotatedVector(), 24
printTextProperties(), 69
printVector2D(), 58
rotateVector(), 27
runLengthCoding(), 37, 38
scalarProduct(), 24, 31
scaleVector2D(), 58, 59
scanBinaryInteger(), 17
scanf(), 16, 25, 61, 64, 65, 68, 71–73
scanf_s(), 73
setBinaryPixel(), 39
sine(), 72, 73
sortDescending(), 62, 63
standardDeviation(), 23
strtok(), 70
substring(), 29
swap(), 64, 65
throwDice(), 17
timeFromSec(), 33
timePeriod(), 33
toInteger(), 28
tone2FrequencyHz(), 19, 21
transposeNxN(), 27
vectorLength(), 58

Geographic coordinate, 47
  Structure, 53
Geographic distance
  Global, 48
  Local, 48
  Maximum, 51

Heap, *see* Memory
Hemisphere, 49

Image
  Binary, 38
  Color, 22, 34
  Pixel, 22
  Region, 38
Integer division, 13
ISO sensitivity, 31

Keyboard input, 12, 16, 44
Keyword
  break, 16, 61, 72
  const, 69, 72
  continue, 16
  default, 69
  do/while, 14, 16, 61
  false, 33, 61
  for, 14, 16, 45, 64
  if, 14
  int, 16
  NULL, 70
  sizeof, 30
  switch, 16, 32, 61, 69, 72
  true, 49, 64
  while, 14, 16, 64

Latitude, 47
Leap year, 14
Leibnitz series, *see* $\pi$
Letter, *see* Character
Linkage, 30
Linked list, 35
Linker, 10
Logic level, 13
Longitude, 47

Macro, 41
Matrix
  Multiplication with vector, 23
  Random numbers, 30
  Rotation, 24
  Transposed, 27
Median filter, *see* Filter
Memory
  Heap, 30
  Stack, 30
Memory management, 29
Modulo operator, 44
Music
  C major scale, 19
  Note, 19, 21
  Octave, 19
  Pitch, 19

Newton's method, *see* Zero-crossing
Noise reduction, 24
Numeric range
  Integer, 14

Object detection, 38
Obstacle detection, 32
Octave, *see* Music
Operating system
  Linux, 10
  Windows, 10
Operation
  Algorithmic, 43

Parameter
  Formal, 19
  Name, 19
Photo, 31
PIN, 16

Pixel, *see* Image
Pointer, 26
    Data type, 28
Power, *see* Electric power
Prime number, 58

Quicksort, *see* Sorting

Random number, 17
    Distribution, 21
Resistor, 11, 13
    Parallel, 11
    Series, 11
Robotics, 32
Route length, 50
Run length encoding, *see* Coding

Scalar product, *see* Dot product
Scope
    File, 30
    Variable, 30
Score board, 22
Seating plan, 54
Sensor
    Distance, 32, 33
    Ultrasonic, 32
Series expansion, 15, 72
Soccer team, 32
Software quality, 43, 76
Sorting, 35
    Bubble sort, 35
    Quicksort, 35
Stack, *see* Memory
Standard deviation, 23
Statement, 10
String, 25
    Parse binary, 37
    Parse character, 25, 28
    Parse integer, 16, 28
    Parse key/value, 25
    Parse words, 28
    Substring, 29
Structure, 31
Synthesizer, 19

Temperature
    Centigrade, 12, 17
    Fahrenheit, 12, 17
Text analysis, 69
Time, 33
Time period, 12
Tone, *see* Audio
Travel planning, 52
Trigonometric function, 24

Ultrasonic sensor, *see* Sensor

Vector, 58
    Absolute, 58
    Average, 23
    Dot product, 24, 31, 59
    Multiplication with matrix, 23
    Rotation, 24, 27
    Scalar product, 58
    Standard deviation, 23
    Structure, 31
Visual Studio, 75
    Keyboard shortcuts, 75
    Project, 10
    Solution, 10, 43
    Startup project, 10
Voltage, 11
    Divider, 12

Wave
    Sine, 19
Wheat on chessboard, 45

Zero-crossing
    Bisection method, 15, 18, 26, 29
    Newton's method, 18, 29