Programming expertise - C exercises

Christian Kappel

April 10, 2025

Contents

Planning

Exercises are subject to change or extension before the planned dates. The current schedule is as follows:

- Introduction to C (2025-04-10)
- Data types and control constructs (2025-04-17)
- Functions (2025-04-24)
- Arrays and pointers (2025-05-08)
- More on pointers, trees, linked lists (2025-05-15)
- File I/O (2025-05-25)

You must submit Exercise 2.4.1 by April 24, 2025, at 5:00 AM (CET), as well as one additional exercise marked with an asterisk (*) by May 29, 2025, at 5:00 AM (CET). Please upload your submissions to the respective Homework 1 and Homework 2 folders on Moodle.

Session 1

Introduction to C

1.1 Getting started

Exercise 1.1.1 Hello world!

1. Compile the following source code and run the program.

```c
#include <stdio.h>

int main()
{
  printf("Hello world!\n");
  return(0);
}
```

2. Create a hello-world.exe program file.

Exercise 1.1.2 Errors

1. Identify the error(s) in the following source code. What message do you receive when compiling or linking?

```c
#include <stdio.h>

int main()
{
  printf("Hello world!\n")
  return(0);
}
```

2. Identify the error(s) in the following source code. What message do you receive when compiling or linking?

```c
#include <stdio.h>

int main()
{
  printf(Hello world!\n");
  return(0);
}
```

3. Identify the error(s) in the following source code. What message do you receive when compiling or linking?

```c
#include <stio.h>

int main()
{
  printf("Hello world!\n");
  return(0);
}
```

Exercise 1.1.3 Simple Multiplications

1. Write a program that multiplies three integers (2, 5, and 13) and prints the result.
2. Also print the three integers.

Exercise 1.1.4 Input/output and testing

1. Write a program to read in an integer from stdin and print out its square.
2. Write a program that continuously reads integers from standard input and prints them, until interrupted (e.g., with Ctrl+C).
3. Write a program that reads 15 integers from standard input and prints their squares.
4. Write a program that reads 15 integers from standard input and prints whether each value is positive or negative.

Exercise 1.1.5 Swapping variables

Assign two integer values to two variables.

1. Swap values using a third variable.
2. Swap values without using a third variable.

1.2 General

Exercise 1.2.1 Variables

1. Which of the following are valid C variable names: 9var, 7a, a7, _var_a, case, fi, f_r, f0r?

2. Find a list of C reserved keywords and review your answer to the previous question.

Exercise 1.2.2 Size matters

1. What is the largest C program ever written? Cite your sources.

Session 2

Data types and control constructs

2.1 Data types

Exercise 2.1.1 Integer data type ranges

1. Write a program that displays the minimum and maximum values for the signed and unsigned versions of char, short, int, and long on your system.
2. Write a program that prints the size (in bytes) of the signed and unsigned versions of char, short, int, and long.
3. How can you calculate the maximum representable value of an integer type based on its size?

Exercise 2.1.2 Numerical storage of characters

1. Write a program that asks the user to enter a letter and prints its numerical (ASCII) representation.
2. Modify the program to print out the letter and its numerical representation.

2.2 Control constructs

Exercise 2.2.1 Printing out numbers

1. Write a program to print out numbers from 1 to 100.
2. Write a program that prints the numbers from 1 to 100, separated by commas.
3. Modify the program to print 10 numbers per line.

Exercise 2.2.2 Sum of integers

1. Write a program that asks for integers and sums them until you enter 0.

Exercise 2.2.3 Optimal data type

1. Write a program that determines the most storage-efficient data type for a given integer value.

Exercise 2.2.4 Multiple choice

1. Write a program that displays a multiple-choice question, asks for an answer, and checks whether it is correct.
2. Modify the program to continue asking until the correct answer is entered.
3. Extend the program to present three questions in sequence, allowing one attempt for each, and then display the number of correct answers.

2.3 General

Exercise 2.3.1 C programming for bioinformaticians

1. Identify three C programs you have used as a bioinformatician or computer-savvy biologist.

2. Provide at least one use-case where C programming is useful in bioinformatics.

Exercise 2.3.2 Most used programming languages

State your sources please.

1. What are the overall most used programming languages? Please cite your sources.
2. What are the most used programming languages by bioinformaticians? Please cite your sources.

2.4 Work on a private computer

Exercise 2.4.1 Work on a private computer *(to be handed in)

1. Choose one of the programming exercises from this session. Record a screen video showing how you compile and run the program on your private computer. Also demonstrate the editor you use and briefly explain your source code. You may use Zoom or another screen recording tool.

Session 3

Functions

Exercise 3.0.1 Variable scopes

1. Compile, link, and run the example programs from the "Variable scope" section of the lecture.
2. How can you use a static local variable to keep a running sum of input integers? Write the corresponding function or program.

Exercise 3.0.2 Math functions

1. Write a program that defines functions to calculate the square, cube, and half (division by two) of both integer and float values.
2. Write a function that takes two values and returns the result of dividing the first by the second - but only if the second is not zero.
3. Write a program that asks for numbers until a stop condition is met, and then calculates their average.

Exercise 3.0.3 Recursion

1. Write a function that calculates the factorial of a given integer. What is the largest input value your implementation can handle?
2. Write a recursive function that calculates the specified power of two.
3. Write a recursive function that calculates a given integer raised to a specified power.
4. List additional example problems where recursion is useful.

Exercise 3.0.4 Multiple source files

1. Place your square, cube, and division-by-two functions in a separate source file. Rewrite your program to use them from that file.
2. Write a function (in a separate source file) that returns whether a letter is uppercase or lowercase. Then, write a program that uses this function to read ten letters and count how many are uppercase and how many are lowercase.
3. How can you create an executable using a precompiled version of your function?

Session 4

Arrays and pointers

4.1 Arrays

Exercise 4.1.1 Smiley point collection

1. Create an array to track smiley points for this month. Initialize all values to zero. Allow the user to enter the day of the month and the number of points earned. Print an overview showing points for all days.
2. As above, but print only the days on which the user earned smiley points.
3. Create a two-dimensional array to collect smiley points for each day of the year. Print a tabular overview of the data.

Exercise 4.1.2 Array size

1. Print the size (in bytes) of the following data types on your system: short, int, long, float, and double.
2. What is the size (in bytes) of the array double[100000][1001] on your system?
3. How many elements does the array int[5][7][3][9][4] contain?
4. What is the maximum size of an integer array on your system? Explain your answer.

Exercise 4.1.3 Character arrays

1. How can you determine the length of the following array: char message[] = "Hello World!";?

4.2 Pointers

Exercise 4.2.1 Pointers

1. Create a long variable and assign a pointer to its address.
2. Print out the variable value by dereferencing the pointer.

3. Repeat the previous step, but also print the memory address of the variable.
4. Do the same for an integer array with 23 elements.

Exercise 4.2.2 Array functions

1. Fill an array with random integers. Write a function that returns the maximum value.
2. Fill two arrays with random floating-point numbers. Write a function that returns the minimum value from each.
3. Write a function that takes two integer arrays of the same size, adds the elements at corresponding positions, and stores the results in a third array.

Exercise 4.2.3 Smiley point collection

1. Write a function that collects smiley points in a two-dimensional array for the entire year.

2. Write another function to print out the tabular overview.
3. Write a function that prints only the days on which smiley points were collected.

Session 5

More on pointers, trees, linked lists

5.1 More on pointers

Exercise 5.1.1 Calculating class average

1. Write a program with a function that asks the instructor for the number of students who participated in the exam.
2. Add a function that collects the students' marks one by one.
3. Add a function that calculates the class average.
4. Add a function that prints the class average.
5. Place each function in a separate source file. Compile and link them using a Makefile.

Exercise 5.1.2 Student list

1. Write a program that stores student names, surnames, and marks in arrays.
2. Add a function to print the data to standard output in a tabular format.
3. Also include the average mark at the bottom of the tabular output.

5.2 Trees, graphs, linked lists

Exercise 5.2.1 Linked lists

1. Write a program that successively fills a linked list with student names, surnames, and grades entered by the user.
2. Print the list in a tabular format, including the average mark.
3. How could you implement sorting for the linked list?
4. List at least three real-world examples where linked lists are used.

Exercise 5.2.2 Binary trees

1. Create the following binary tree:

```
     2
    / \
   9   3
  / \   \
 7   4   99
```

2. Write a function that calculates the size of the tree (i.e., the number of nodes).
3. Write functions to find the minimum and maximum values in the tree.
4. Find three or more examples where binary trees are used.

Exercise 5.2.3 Graphs *(may be handed in)

1. How can you represent the graph shown in Figure 5.1 in code?
2. Write a program that implements this graph representation.
3. Add a function that calculates the average number of neighbors per vertex.

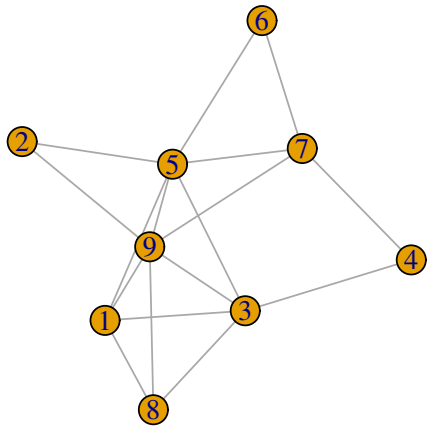

Figure 5.1: Simple undirected graph.

Session 6

File I/O

Exercise 6.0.1 Writing numbers to a file

1. Write a program that includes three functions to calculate the square, cube, and square root of a given number.
2. Use these functions to calculate the square and square root of integers from 0 to 1000, and write the output to a file.
3. Modify the program to accept user input until a stop condition is met. Write the numbers, their squares, cubes, and square roots to a user-defined file.
4. Modify the program to prompt the user to either create/overwrite or append the output file.

Exercise 6.0.2 Formatted Input/Output

1. List and describe the format specifiers used in printf/fprintf.
2. List and describe the format specifiers used in scanf/fscanf.
3. Which format specifier would you use to print 1500000 in scientific notation to a file?

Exercise 6.0.3 Student list

1. Modify your program that collects student names, surnames, and grades to save the output to a user-defined file.
2. Add statistics at the bottom of the output file, including: best and worst mark, average, median, total number of students, number of successful students, and the success rate.

Exercise 6.0.4 Reading, sorting, writing *(may be handed in)

1. Create a dataset containing names, family names, and grades for at least 348 imaginary students. The grades should roughly follow a normal distribution. You do not need to use C for this step.
2. Read the data and store it in a linked list.
3. Sort the list by family name and write it to a file.

4. Then sort the list by grade and write it to a separate file.

Exercise 6.0.5 Binary vs text

1. Create a two-dimensional array filled with random integer values.
2. Write the array to a text file and then read the data back.
3. Write the array to a binary file and then read the data back.
4. Compare both approaches in terms of speed and file size.

Bibliography