KTH Information and Communication Technology

IS1200/IS1500

Lab2 – C Programming
2024-07-30
v1.2

Introduction

Welcome to the second lab in the course. In this lab, you will learn the basics of the C programming language. More specifically, after finishing this lab, you will be able to

1. Construct short C programs that include for-loops, if-statements, arrays, and pointers.
2. Analyze C programs and understand how a compiled program is laid out in memory.

Preparations

You must do the lab work in advance, except for a few specific tasks. The teachers will examine your skills and knowledge at the lab session.

Assignments 1 through 5 must be prepared completely before the start of your lab session.

Assignments 6 (the surprise assignment) will be performed during your lab session.

You can ask for help any time before the lab session. There are several ways to get help; see the course website for details and alternatives.

During the lab session, the teachers will perform examinations, but can also offer help. Make sure to state clearly that you want to **ask questions**, rather than being examined.

Sometimes, our teachers may have to refer help-seekers to other sources of information, so that other students can be examined.

NOTE: In this exercise, you MUST explicitly declare in the code whether you or your lab partner wrote the code. That is, you should explicitly state who typed on the computer. It is NOT allowed to copy ANY code from anyone else or from the Internet.

Out of the four first assignments, each of you (lab partners) should author two assignments each. This means that it is not allowed for just one of you to sit and type the code for all the assignments. You should still sit together and discuss all the problems and the solutions and the lab partner who is not typing should still be able to analyze the program and explain all the details of the solution.

It is also allowed that both of you create your own solutions, for all exercises. This is actually the best alternative. It is OK if your solutions are similar, as long as you have written them individually.

Resources

For Assignments 1 to 4, you may use a C compiler of your choice. We recommend using compiler such as `gcc` from the command line interface. See the course website for more information about

lab resources.

For assignment 5, you will use a DTEK-V development board, which includes a RISC-V/32 processor. To access the lab environment, either using your own computer or through KTH computers, please visit the "Software Resources" page on Canvas.

The files for assignments 1-4 can be downloaded from Canvas (*Laboratory Exercises*, bottom of the page). There are three C files: `prime.c, print-primes.c and sieves.c`.

Reading Guidelines

Review the following course material while you are preparing your solutions.

- Lectures 1 and 4.
- The Harris & Harris (2021) book, Appendix C on C Programming.
- The online C programming tutorial links are available on the course webpage under *Literature and resources*.

Examination

The examination is grouped into parts. Each part is usually a group of several assignments. Examining one part takes 5–15 minutes. Make sure to call the teacher immediately when you are done with an assignment.

Part 1 – Assignments 1 to 3.

Part 2 – Assignment 4.

Part 3 – Assignments 5 and 6.

The teacher checks that your program behaves correctly, that your code follows all coding requirements, and that it is easily readable. **The teacher will also ask you to compile your program. If there are any errors or warnings, the teacher will ask you to solve these problems before the examination begins.**

The teacher will also ask questions, to check that your knowledge of the design and implementation of your program is deep, detailed, and complete. When you write code, make detailed notes on your choices of algorithms, data-structures, and implementation details. With these notes, you can quickly refresh your memory during the examination, if some detail has slipped your mind.

Assignments

Assignment 1: Basic Control-Flow

The purpose of this assignment is to learn how to write a simple C program that includes basic control structures, including if-statements and for-loops.

Assuming that you are using gcc as your compiler, compile and run the file as follows:

```
gcc prime.c -o prime
./prime
```

The program prints the number 0 three times.

Task: Your task is to implement the body of function `is_prime()`, so that it returns value 1 if parameter `n` is a prime number, and 0 if it is not. The function must behave correctly for all positive integer numbers. The function should not use data structures, only simple loops and conditional `if`-statements.

Assignment 2: Functions and Side Effects

The purpose of this assignment is to learn the concepts of functions, arguments, global variables, and side effects.

The function `print_primes` includes example code that shows how to print a list of prime numbers to the standard output.

Task: You should now do the following:

1. Create a new function called `print_number` that should have one integer parameter value representing the number that should be printed. If the function is called several times in a row, the output should be printed in several columns, as shown in the example template code. The number of columns is defined by the `COLUMNS` macro. The function should not return any value.
2. Insert function `is_prime` from the previous exercise. Create a new body for the function `print_primes`. The function should print all prime numbers from 2 to `n` by calling functions `print_number` and `is_prime`.

If you execute the program as follows:

```
./print-primes 105
```

the output should be all prime numbers up to and including 105. That is, the output should be as follows:

```
  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
```

Questions for Assignment 2

Before the oral exam, you should prepare the answers to the following questions. You will need to be able to answer these questions to pass the assignment.

- What does it mean when a function does not return a value? How do you state that in a program? How can the function (or more precisely, the procedure) perform anything useful?

- How did you implement the side effect needed to make `print_number` behave correctly?

Assignment 3: Arrays

The purpose of this assignment is to learn how to use arrays in a program.

Task 1: Your task is now to create a new program called `sieves.c`. This program file should use the main function and the `print_number` function from the previous exercise. However, instead of using the `print_prime` function, a new function called `print_sieves` should be implemented. This function should have one parameter that states the max prime number values. This program should give the same output for a given input as in Assignment 2, but the `print_sieves` function should implement the algorithm *Sieve of Eratosthenes*. See the Wikipedia page for an explanation of this old, simple, yet powerful algorithm: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes.

In task 1, you should allocate all data on the stack, using a local array declaration. You must not allocate more memory than necessary. In this case, the memory consumption should not exceed (at most) `n+8` bytes, where `n` is the parameter value in function `print_sieves`.

Task 2: Copy the content of `sieves.c` to a new file `sieves-heap.c`. Now, allocate the array data on the heap instead of the stack. Do not forget to release the memory when you no longer need it anymore.

Questions for Assignment 3

Before the oral exam, you should prepare the answers to the following questions. You will need to be able to answer these questions to pass the assignment.

- How did you represent the marking of 'prime' and 'not a prime' in the memory array?
- What are the main steps in the algorithm? How have you implemented these steps?
- What is the largest prime number that you can print within 2 seconds of computation? What is the largest number you can print within 10 seconds? Is it the same for `print_prime.c`, `sieves.c`, and `sieves-heap.c`? Why or why not?

Assignment 4: Pointers

The purpose of this exercise is to get a detailed understand of pointers and pointer concepts such as dereferencing and pointer arithmetics.

Locate the files `pointers.c` from the course website. Note that this is just a template and cannot be compiled without adding additional code. Also locate the file `pointers.S`. The assembly file is complete and may be executed in the RARS simulator.

Task: The file `pointers.S` contains an assembler program with two functions: `work` and `copycodes`. Your task is to translate these two assembly functions into C functions. Note that the data declarations for `text1` and `text2` have already been provided. You must also declare `list1`, `list2`, and `count` correctly in the C code. Your C functions must implement pointers and arguments correctly, including correct handling of types. You may not use array indexing in this exercise.

> **Note:** Your two C functions (`work` and `copycodeS`) must be interchangeable with the assembly language versions. Parameters and results must match, so that the code to call your function is the same as the code to call the corresponding assembly-language function.

If your program is implemented correctly, the following should be printed when executing the program `pointers.c`:

```
list1: ASCII codes and corresponding characters.
```

```
0x054 'T' 0x068 'h' 0x069 'i' 0x073 's' 0x020 ' ' 0x069 'i' 0x073 's'
0x020 ' ' 0x061 'a' 0x020 ' ' 0x073 's' 0x074 't' 0x072 'r' 0x069 'i'
0x06E 'n' 0x067 'g' 0x02E '.'

list2: ASCII codes and corresponding characters.
0x059 'Y' 0x065 'e' 0x074 't' 0x020 ' ' 0x061 'a' 0x06E 'n' 0x06F 'o'
0x074 't' 0x068 'h' 0x065 'e' 0x072 'r' 0x020 ' ' 0x074 't' 0x068 'h'
0x069 'i' 0x06E 'n' 0x067 'g' 0x02E '.'

Count = 35

Endian experiment: 0x23,0x00,0x00,0x00
```

Questions for assignment 4

Before the oral exam, you should prepare the answers to the following questions. You will need to be able to answer these questions to pass the assignment.

- Explain how you get the pointer addresses to the two char arrays (`text1` and `text2`) and the counter variable (`count`) in function `work()`.
- What does it mean to increment a pointer? What is the difference between incrementing the pointer that points to the ASCII text string, and incrementing the pointer that points to the integer array? In what way is the assembler code and the C code different?
- What is the difference between incrementing a pointer and incrementing a variable that a pointer points to? Explain how your code is incrementing the `count` variable.
- Explain a statement in your code where you are dereferencing a pointer. What does this mean? Explain by comparing it with the corresponding assembler code.
- Is your computer using big-endian or little-endian? How did you come to your conclusion? Is there any benefit of using either of the two alternatives?

Assignment 5: Memory Layout

The purpose of this task is to get a good understanding of how data and code are allocated in the memory. This includes local variables, global variables, pointers, and arrays.

Download and unzip the file `riscv32tests.zip` from the course website. This assignment uses the DTEK-V toolchain and the DTEK-V board. Make sure to also download `time4riscv.zip` from the previous lab. Unzip all the content into a single directory and run `make` and `dtekv-run main.bin` to upload the program to the board. Consult the lecture slides (or Canvas page) to see the memory layout of the binary.

Task: Your task is to inspect the file `main.c` and to browse through the information that is shown on your host monitor. In the `main` function, a function `print_byte/word(string, addr)` is called 20 times. This help function saves a string name `str` and displays this on your local monitor. Together with the text, the address `addr` (parameter two of `print_byte/word`) is also displayed, together with the data word (32 bits) that the address `addr` points to. You should now go through all the 20 different items that are displayed, inspect the code, and be ready to answer the following questions:

Questions for Assignment 5

Before the oral exam, you should prepare the answers to the following questions. You will need to be able to answer these questions to pass the assignment.

- Consider `AM18`, `AM19`, and `AF1`. Explain why `gv` ends up with the incremented value, but `m` does not.
- Pointer `cp` is a character pointer that points to a sequence of bytes. What is the size of the `cp`

pointer itself?

- Explain how a C string is laid out in memory. Why does the character string that `cp` points to have to be 9 bytes?
- Which addresses have `fun` and `main`? Which sections are they located in? What kind of memory are they stored in? What is the meaning of the data that these symbols points to?

Before the examination, you should also try to answer the following. When the lab-assistant performs the examination, he/she can also clarify anything that you did not understand with the following questions:

- Which addresses are variables `in` and `gv` located at?
- Variables `p` and `m` are not global variables. Where are they allocated? Which memory section is used for these variables? Why are the address numbers for `p` and `m` much larger than for `in` and `gv`?
- At print statement `AM5`, what is the address of pointer `p`, what is the value of pointer `p`, and what value is pointer `p` pointing to?
- At print statement `AM7`, what is the address of pointer `p`, what is the value of pointer `p`, and what value is pointer `p` pointing to?
- Consider `AM14` to `AM17`. Is the RISC-V processor using big-endian or little-endian? Why?

Assignment 6: Surprise assignment

You will receive a surprise assignment at the lab session. We will hand out the surprise assignment directly at the beginning of the lab, presupposed that you have finished all the other assignments. If you have not received the surprise assignment, please ask the teaching assistant for the assignment paper.

Version History

1.0 – First published version. 2015-09-25.

1.1 – Added a short note that all programs need to be error and warning free, before examination. 2017-09-04.

1.2 – Ported to RISC-V/32 platform. 2024-07-27