KTH Information and
Communication Technology

IS1200/IS1500

Lab2 – C Programming

Version 1.1 2015-09-17

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 shorter C programs that includes 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 4 must be prepared completely before the start of your lab session.

Assignments 5 and 6 can be performed during your lab session. However, it is strongly recommended that you also go through assignment 5 before the lab session, but it is not compulsory.

You can ask for help anytime 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 work with examination, 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 if you or your lab partner wrote the code. That is, you should explicitly state who actually 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 that just one of you are sitting and actually typing the code for all 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.

Lab Environment

For Assignments 1 to 4, you may use a C compiler of your choice. We recommend that you use a 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 the MCB32 Tool together with the ChipKit Uno32 board that you received in lab 1.

Reading Guidelines

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

- Lectures 1 and 4.
- The Harris & Harris (2013) book, Appendix C on C Programming.
- The online C programming tutorial links that are available on the course webpage under "Literature".

Examination

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 – Assignments 4.

Part 3 – Assignment 5 and 6.

The teacher checks that your program behaves correctly, and that your code follows all coding requirements and is easily readable.

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 examination, if some particular 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.

Download file `prime.c` from the course web page. 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 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 correct for all positive integer numbers. The function should not use any 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.

Download file `print-primes.c` from the course web page. Function `print_primes()` includes example code that shows how a list of prime numbers should be printed 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 define `COLUMNS`. 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`, so that it prints `n` number of primes 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 between 1 and 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

These questions are useful to prepare yourself for the oral examination. However, at the examination, the teacher may choose to ask questions that are not on this list.

- What does it mean when a function does not return a value? How do you state that in a program? How can then the function (or more precisely, the procedure) perform anything useful?
- How did you implement the side effect that is needed to make `print_number` behave correct?

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` functions 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 the same output for a given input as in Assignment 3, but the `print_sieves` function should implement the algorithm *Sieve of Eratosthenes*. See the Wikipedia page for an explanation of this old and 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 be more than `n` byte, 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 on the stack. Do not forget to release the memory when you do not need to use it anymore.

Questions for assignment 3

These questions are useful to prepare yourself for the oral examination. However, at the examination, the teacher may choose to ask questions that are not on this list.

- How did you represent the marking of 'prime' and 'not a prime' in the memory array?
- Which 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.

Download file `pointers.c` from the course website. Note that this is just a template and cannot be compiled without adding additional code. Download `pointers.asm`. The assembly file is complete and may be executed in the MARS simulator.

Task: The file `pointers.asm` 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` is already provided, but you also need to 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.

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

These questions are useful to prepare yourself for the oral examination. However, at the examination, the teacher may choose to ask questions that are not on this list.

- 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 you 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 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 with 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 is allocated in the memory. This includes local variables, global variables, pointers, and arrays.

Download and unzip file `uno32tests.zip` from the course website. This assignment uses the MCB32 tool chain and the ChipKIT board. Please run `make` and `make install` to upload the program on the board. As part of the exercise, consider the memory map for PIC32, described in the PIC32 Family Reference Manual, Section 3, page 12: http://ww1.microchip.com/downloads/en/DeviceDoc/60001115H.pdf. The virtual memory map is the addresses that are visible to the programmer and the processor. In the figure, addresses for different memory sections are given.

Task: Your task is to inspect the file `main.c` and to browse through the information that is shown on the ChipKIT display. In the `main` function, a function `saveword(string, addr)` is called 20 times. This help function saves a string name `str` and displays this as a separate page on on the ChipKIT display. Together with the text, the address `addr` (parameter two of `saveword`) is also displayed, together with the data word (32 bits) that the address `addr` points to. You can swithc pages on the ChipKIT board by clicking the buttons. 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

These questions are useful to prepare yourself for the oral examination. However, at the examination, the teacher may choose to ask questions that are not on this list.

- Which addresses are variables `in` and `gv` located at? Which memory sections according to the PIC32 memory map? Why?
- 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?
- 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?
- Pointer `c` is a character pointer that points to a sequence of bytes. What is the size of the `c` pointer itself?
- Explain how a C string is laid out in memory. Why does the char string that `c` points to have to be 4 bytes?
- Consider `AM14` to `AM17`. Is the PIC32 processor using big-endian or little-endian? Why?
- Consider `AM18`, `AM19`, and `AF1`. Explain why `gv` ends up with the incremented value, but `m` does not.

Assignment 6: Surprise assignment

You will get a surprise assignment at the lab session.

This assignment must be completed during the session.

Revision History

- 2015-09-15: First version.
- 2015-09-17: Updated the example printout in Assignment 4. The characters output was incorrect. This update does not change the assignment itself.