



Sami Ilvonen
Petri Nikunen



Introduction to HPC with C

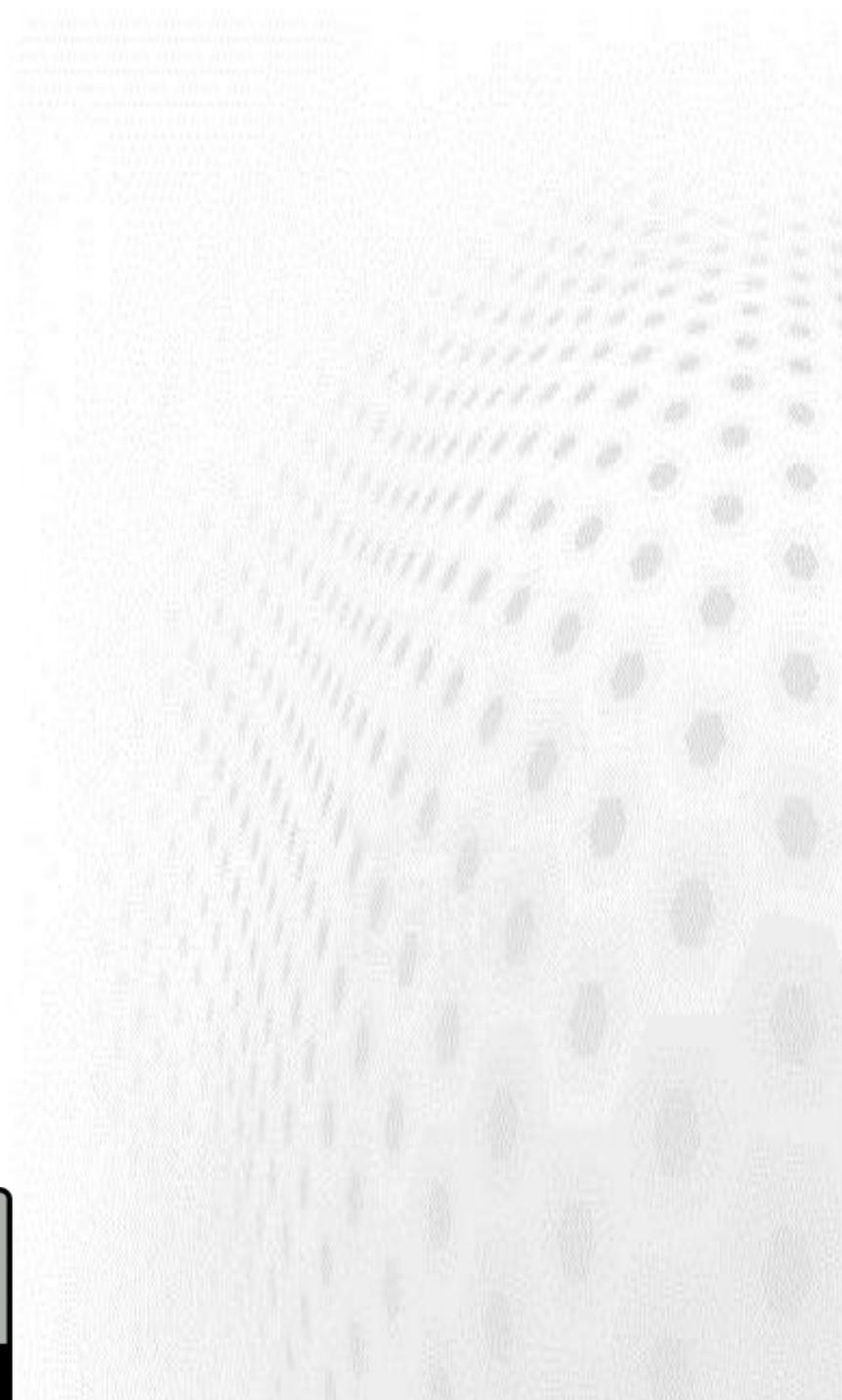
Sep 7 – 9, 2016

@ CSC – IT Center for Science Ltd, Espoo

```
int **b1, **b2;

/* Initialise metadata */
board_1->height = height;
board_1->width = width;
board_2->height = height;
board_2->width = width;

/* Allocate memory for grids */
board_1->space = malloc((height + 2) * sizeof(int *));
board_1->space[0] = malloc((height + 2) *
                          (width + 2) * sizeof(int));
for(i = 1; i < height + 2; i++) {
    board_1->space[i] = board_1->space[0] + i * (width + 2);
}
board_2->space = malloc((height + 2) * sizeof(int *));
board_2->space[0] = malloc((height + 2) *
                          (width + 2) * sizeof(int));
for(i = 1; i < height + 2; i++) {
    board_2->space[i] = board_2->space[0] + i * (width + 2);
}
```



All material (C) 2016 by CSC – IT Center for Science Ltd.
This work is licensed under a **Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported** License, <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Agenda

Wednesday

09:00-09:45 **Introduction**

09:45-10:00 Coffee break

10:00-10:45 **Getting started with C**

10:45-12:00 **Exercises**

12:00-13:00 Lunch break

13:00-14:00 **Control structures and loops**

14:00-14:30 **Exercises**

14:30-14:45 Coffee break

14:45-15:45 **Exercises**

15:45-16:00 Wrap-up day 1

Thursday

09:00-10:00 **Pointers and dynamic memory management**

10:00-10:15 Coffee break

10:15-11:00 **Exercises**

11:00-11:30 **Strings, structures and datatypes**

11:30-12:00 **Exercises**

12:00-13:00 Lunch break

13:00-14:00 **Exercises**

14:00-14:30 **Dynamic arrays**

14:30-14:45 Coffee break

14:45-15:45 **Exercises**

15:45-16:00 Wrap-up day 2

Agenda

Friday

09:00-09:45 **I/O**

09:45-10:00 Coffee break

10:00-11:15 **Exercises**

11:15-12:00 **Code structuring**

12:00-13:00 Lunch break

13:00-14:00 **Exercises**

14:00-14:15 Coffee break

14:15-15:00 **Programming practices and debugging**

15:00-15:45 **Exercises**

15:45-16:00 Course wrap-up

Introduction to HPC with C – exercises

1. Basics, compiling, datatypes and expressions

- a) Write from a scratch a short program that prints out a sentence (e.g. “hello, world!”). Compile the code using GNU C compiler (gcc) and execute the program. See the lecture material for relevant code examples and compiler commands.
- b) Perform simple expressions with different types of variables (integers, floats, doubles) and print out results with **printf** using correct format. File `ex1b_expressions.c` contains a skeleton code to start with.
- c) Add some pointer variables to the previous exercise and perform some expressions via the pointers. Investigate both the original variable and the pointer variable after expressions. A skeleton code is provided in `ex1c_pointers.c`

2. Control structures

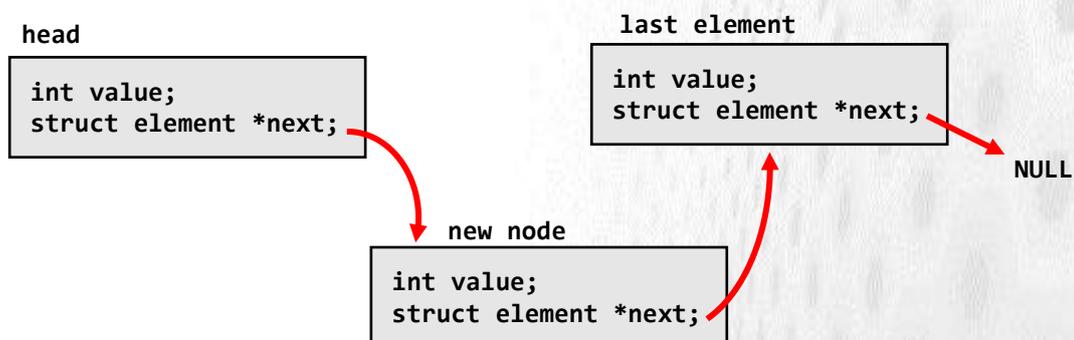
- a) Write a control structure which checks whether an integer is negative, zero, or larger than 100 and performs corresponding **printf**. Investigate the behavior with different integer values. Skeleton code is given in file `ex2a_if_else.c`.
- b) In the file `ex2b_my_pow_prog.c` you find a skeleton code for a program that calculates powers. Finish the code by adding a missing update loop.
- c) Fibonacci numbers are a sequence of integers defined by the recurrence relation
$$F_n = F_{n-1} + F_{n-2}$$
with the initial values $F_0=0$, $F_1=1$. Print out Fibonacci numbers $F_n < 100$ using a **while** loop.
- d) Write a function that determines if a given character is a vowel or a consonant. Use the switch-case structure. Skeleton code is given in file `ex2d_switch.c`.

3. Pointers and dynamic memory management

- a) Implement a function that takes three arguments of type double. The function should return in the third argument the sum of the first two arguments. Template code is in file `ex3a_pointers.c`.
- b) Modify the given template `ex3b_pointer_swap.c` and implement a function that swaps its pointer arguments **a** and **b** so that they point to **b** and **a** in reversed order. Can you make a version that can accept all pointer types? (Hint: remember the type **void**).

4. Strings, structures and datatypes

- a) Implement a program that converts the strings given as command line arguments to upper case equivalents and prints them out on separate lines. Template can be found in `ex4a_upcase.c`. Use can use the routine **toupper** defined in the header file `ctype.h` to convert the string letter by letter. (Hint: **man 3 toupper**).
- b) Modify the given template (`ex4b_structs.c`) so that you use a structure to pass the values to the output routine that prints the basic information of a book.
- c) Implement the insertion routine for the linked list example program given in the template file `ex4c_linked_list.c`. Note that the list is implemented so that empty list has two nodes: **head** defined in the main routine holds the starting point and **init_list** routine allocates one place holder for the end point of the list. Implement the insert so that new value is added to the beginning of the list. After insertion the **head->next** should point to the new node.



5. Dynamic arrays

- a) Write a function that allocates an integer array and sets the values in the array from 1 to the N where N is the size of the array given as an input argument. Print out the values in the calling main function. Skeleton is in file `ex5a_malloc.c`.
- b) Extend the allocation function so that it allocates a two-dimensional array. Implement also a routine that frees the array. Skeleton file `ex5b_array.c` has some utility routines that print out the array for checking.

6. I/O

- a) Implement a program that asks for user input and echoes back all numerical characters (0-9) and drops out all other input. If no numbers is given, program should report that no numbers were given. Skeleton code is in file `ex6a_echo.c`.
- b) Extend the skeleton file `ex6b_file_io.c` file and implement a program that reads the input file "testdata.dat" line-by-line, converts the read text to upper case and prints the results to standard output.

7. Code structuring

- a) Start from the provided single file version of a heat solver and move the function `save_png` to a separate file `pngwriter.c`. Create a header file `pngwriter.h` with needed declarations. Try to compile the new version and check that it works. Note that you will need two libraries, `libpng` and `libm` (use compiler flags `-lpng` and `-lm`).
- b) Split the solver further so that the main routine is in a separate file `ex7_main.c` and the needed declarations in a header file `ex7_heat_solver.h`.

8. Debugging

Take a look at the demo code in folder `demo`, compile it using the provided `makefile` (issue the command `make`) and run the program. Find the reason for segmentation fault using `gdb`, fix the problem and check that the program works correctly after your corrections.