C Programming

Absolute basics

- languages: natural and artificial
- machine languages
- high-level programming languages
- obtaining the machine code: compilation process
- recommended readings
- your first program
- variable – why?
- integer values in real life and in “C”, integer literals

Data types

- floating point values in real life and in “C”, float literals
- arithmetic operators
- priority and binding
- post- and pre -incrementation and -decrementation
- operators of type op=
- char type and ASCII code, char literals
- equivalence of int and char data
- comparison operators
- conditional execution and if keyword
- printf() and scanf() functions: absolute basics

Flow control

- conditional execution continued: the “else” branch
- more integer and float types
- conversions – why?
- typecast and its operators
- loops – while, do and for
- controlling the loop execution – break and continue
- logical and bitwise operators

Arrays

- switch: different faces of ‘if’
- arrays (vectors) – why do you need them?
- sorting in real life and in a computer memory
- initiators: a simple way to set an array
- pointers: another kind of data in “C”
- an address, a reference, a dereference and the sizeof operator
- simple pointer and pointer to nothing (NULL)
- & operator
- pointers arithmetic
- pointers vs. arrays: different forms of the same phenomenon
- using strings: basics
- basic functions dedicated to string manipulation

Memory management and structures

- the meaning of array indexing

- the usage of pointers: perils and disadvantages
- void type
- arrays of arrays and multidimensional arrays
- memory allocation and deallocation: malloc() and free() functions
- arrays of pointers vs. multidimensional arrays
- structures – why?
- declaring, using and initializing structures
- pointers to structures and arrays of structures
- basics of recursive data collections

Functions

- functions – why?
- how to declare, define and invoke a function
- variables' scope, local variables and function parameters
- pointers, arrays and structures as function parameters
- function result and return statement
- void as a parameter, pointer and result
- parameterizing the main function
- external function and the extern declarator
- header files and their role

Files and streams

- files vs. streams: where does the difference lie?
- header files needed for stream operations
- FILE structure
- opening and closing a stream, open modes, errno variable
- reading and writing to/from a stream
- predefined streams: stdin, stdout and stderr
- stream manipulation: fgetc(), fputc(), fgets() and fputs() functions
- raw input/output: fread() and fwrite() functions

Preprocessor and complex declarations

- preprocessor – why?
- #include: how to make use of a header file
- #define: simple and parameterized macros
- #undef directive
- predefined preprocessor symbols
- macrooperators: # and ##
- conditional compilation: #if and #ifdef directives
- avoiding multiple compilations of the same header files
- scopes of declarations, storage classes
- user -defined types – why?
- pointers to functions
- analyzing and creating complex declarations