Topics for Final Test

1. Fundamentals
    - Interpreter, compiler, preprocessor, assembler, linker
    - Data types:
        - integer: signed and unsigned, `char`, `short`, `int`, `long`, `long long` data types
        - floating-point: `float`, `double`, `long double`
    - Preprocessor directives: `include` and `define`
    - Lexical conventions and tokenization: identifiers, keywords, constants, operators, tokens, whitespace
        - How many tokens in `printf("Average value: %f\n", average);`
    - Understand difference between variables and constants
    - Understand declarations and definitions
2. Creating a simple program
    - source file consists of functions
    - understand meaning of function declarations (prototypes) and definitions
    - role of function `main`
    - understand purpose of header files
    - what is a function argument and a function parameter
    - functions consist of statements which in turn consist of expressions
    - single- and multi-line comments
    - declaration and definition of variables
    - what does compiling mean?
    - what does linking mean?
    - understand options of `gcc`: `-std=c11`, `-pedantic-errors`, `-Wstrict-prototypes`, `-Wall`, `-Wextra`, `-Werror`, `-c`, `-o`
    - understand various stages of compiler driver [preprocessor, compiler proper, assembler, linker]
    - understand how to use `gcc` flags to invoke only the preprocessor, compiler, assembler
    - understand diagnostic warning and error messages from the compiler
    - I/O functions in standard library
    - mathematical functions in standard library
    - linking with external libraries
3. Arithmetic and assignment expressions
    - operators, operands
    - arithmetic operators
        - multiplicative operators
        - additive operators
        - reinforce notion of type and behavior of `/` and `%` operators
    - expression evaluation using expression trees
    - precedence
    - associativity
    - assignment and side-effects

- *lvalues* and *rvalues*
  - what is *lvalue* and what is *rvalue*?
  - why does `x = 7` work and not `7 = x`
- compound assignment operators
- `sizeof` operator
- type cast operator
- implicit type conversion

4. Conditionals
   - relational operators [`<`, `<=`, and so on] and equality operators: `==`, `!=`
     - integer values for true and false expression evaluations
     - applications and examples
   - boolean values resulting from relational expressions
   - `_Bool` type and `<stdbool.h>` header and `bool`, `true`, and `false` macros
   - logical operators (`&&`, `||`, `!`)
     - order of operand evaluation and short-circuit evaluation
     - precedence and associativity
     - applications and examples
   - `if` statement
     - meaning of a statement [`;` or `expr;` or block of statements delimited by `{` and `}`]
     - `else` clause
     - nested `if` statements
     - dangling `else` problem
     - applications and examples
   - Conditional operator `? :`
   - `switch` statement
     - what is a label?
     - `case` keyword
     - `break` keyword
   - Why these operators are special (hint: they are sequence operators): logical or: `||`, logical and: `&&`, conditional operator: `? :`

5. Iteration - looping and repetitions
   - `while` statement
     - different problem solving techniques involving counters, sentinels
     - introduce `getchar` and `putchar` standard library functions
     - problem solving techniques involving text files: emulating Unix programs such as `wc`, `cp`, `cat`, ...
   - `for` statement and significance of its three expressions
   - rewriting a `for` statement as a `while` statement and vice-versa
   - `do while` statement
   - guidelines on when to use which iteration statement
   - infinite loops
   - Jump statements: `break`, `continue`, and `return` statements

- increment and decrement operators

6. Formatted input/output
    - abstraction of I/O using streams
    - `stdin`, `stdout`, and `stderr`
    - `printf` function: printing integers and floating-point numbers to `stdout`
    - `scanf` function: reading integers and floating-numbers from `stdin`, significance of `&` operator in arguments
    - conversion specifiers used in `printf` and `scanf` functions to print common integer and floating-point types
    - controlling width and precision of output of integers and floating-point numbers
    - escape sequences [able to understand the meaning of this: `printf("\"\\%%\t%%\t%%\\\"\n");`]
7. Data Types
    - Integral types and their relative sizes
    - Floating point types and their relative sizes
    - Literal constants
    - `const` type qualifier for variables
8. Functions
    - Function declarations [or prototypes]
    - Function definitions
    - Calling functions: what is function call expression, what are arguments, what are parameters
    - Functions that return a value
    - Functions that do not return a value (`void`)
    - Functions that take parameters
    - Functions that do not take any parameters (`void`)
    - `return` statements (single and multiple returns)
    - Significance of function ordering in a program
    - Problems with missing function prototypes when compiling
    - Problems with missing functions definitions when linking
    - C's "pass-by-value" mechanism
    - Passing parameters using the stack
    - Functions: storage duration, scope and linkage
9. Larger programs with multiple source files
    - Programs consisting of multiple source files
    - headers files, function declarations [or prototypes], function definitions
    - idea of compiling individual source files into object files and linking these object files plus external libraries into single executable
    - Storage duration [lifetime]: automatic vs static storage duration
    - Scope [visibility]: internal vs external variables
    - Linkage [accessibility]: no linkage, internal linkage, external linkage
    - Declarations of variables at file scope [global variables] and local scope
    - Use and purpose of storage specifiers: `auto`, `register`, `static`, `extern`
    - One definition rule

    - What are the different regions of a executable program's memory image? You should know about what the `.text`, `.rodata`, `.data`, `.bss`, `stack`, and `heap` sections contain.
    - You should be able to answer the following questions:
        - How can variables that are defined in one source file be referenced and shared by authors of other source files?
        - What happens if multiple source files declare functions and variables with the same names?
        - What happens if a source file declares multiple variables in different regions of program text with the same name?
        - How can functions defined in one source file be accessible to functions in other source files?
        - How can all the different functions in different source files be connected together to ensure that a executable program is generated?
10. Pseudo-random numbers [first used in programming homework 6]
    - `srand` and `rand` functions and `RAND_MAX` macro from the Standard C library
    - Mapping random numbers to integer and floating-point ranges
11. Arrays
    - Arrays and `sizeof` operator
    - Relationship between pointers and arrays
    - Accessing array elements [to read and write]
    - Looping over arrays
    - Zero-based arrays [first element has index 0]
    - Initializing arrays using curly braces
    - Static arrays vs. dynamically allocated arrays
    - Passing arrays to functions
    - Two-dimensional arrays: Definition; initialization; array of arrays conceptualization; passing two-dimensional arrays to functions
12. Pointers
    - Pointer variables
    - Address of operator
    - Indirection [dereference] operator
    - Passing pointers to functions
    - Returning pointers from functions
    - Relationship between pointers and arrays
    - Pointer arithmetic
    - Compact pointer expressions [combinations of dereference and post- and pre-fix increment and decrement operators]
    - Array of pointers
13. Strings
    - Character arrays vs. strings
    - String literals
    - null (`'\0'`) terminated strings
    - `char` pointers and strings
    - Initializing character arrays and strings
    - Strings and `sizeof` operator
    - Looping over strings and character arrays
    - String input and output

- Common string functions declared in `<string.h>` including `strlen`, `strcpy`, `strcmp`, `strcat`
- Command-line parameters

14. File I/O
    - Opening files, closing files, and position cursor within file: `fopen`, `fclose`, `ftell`, `fseek`
    - Reading and writing text files: `fprintf`, `fscanf`
    - Functions for reading/writing strings: `fgets`, `fputs`
15. Structures
    - Declaring structures
    - Accessing members of structures
    - Structure member operator (`.`) and structure pointer operator (`->`)
    - Data alignment requirements for structure members
    - Nested structures
    - Initializing structures using curly braces
    - Passing structures to functions [by value and address]
    - Returning structures from functions [by value]
16. Dynamic Memory Allocation
    - Allocating memory with `malloc`; you must know that `calloc` and `realloc` are variations of `malloc`- just remember that they exist in case you need more options than those provided by `malloc`
    - Deallocating memory with `free`
    - Problems with memory management
        - Memory leaks
        - Dangling pointers
        - Multiple frees
    - Using Valgrind to check for memory leaks
17. Miscellaneous
    - `typedef` storage specifier for declaring new names for existing types
    - `static` storage specifier for providing lifetime throughout program duration for variables with automatic extent
    - `static` storage specifier for restricting names of global objects [variables and functions] from being exported to the linker
    - `extern` storage specifier for declaring to compiler objects [variables and functions] that are declared in other source files
    - `const` type qualifier for defining read-only variables
    - Enumerations and enumeration constants
    - `assert` macro for debugging
    - Selection sort algorithm
    - Algorithms for processing half-open range of values