Midterm Review

This is a general study guide for the midterm. In addition to the list of topics below, study the class notes and the relevant sections of the textbook. Do not just memorize terms; most questions will be testing your understanding of the concepts. The exam will be closed book/notes and may contain several different kinds of questions (such as multiple choice, short-answer, debugging, coding, etc.). If you have to write any code, it will not be extensive, but you will be graded for syntax and correctness.

- **The compilation process**
  What does the compiler do? What does the linker do? What is the debugger and how can it help us find errors? What are preprocessor directives?

- **Variables & constants**
  Naming rules & conventions: What is a valid variable name? What should be avoided when naming identifiers?

  Literals: What is a literal? Why is better to use `#define` for literals? What is the advantage of `const` over `#define`?

  Basic operators: What is an expression? What are precedence & associativity? What are the precedence and associativity rules for the arithmetic, logical and relational operators?

- **Functions, etc.**
  Parameters: What are the actual parameters? What are the formal parameters? What is a prototype and its syntax? What is call-by-value and what is call-by-reference? When the argument is a pointer is that call-by-value or call-by-reference? Why and how would we use `const` when passing an argument by reference?

  Scope & storage: What does *scope* mean? What is local scope? File scope? Class scope? Prototype scope? What does *storage class* mean? What is automatic storage? Static storage? External storage?

  Stack frame: What is a stack frame?

  Recursion: The basics of recursion. Base case, recursive step.

- **Conditionals**
  `if`*,* `if/else` *and* `switch` *statements*: syntax, use. What should be the type of the control variable in a `switch`? Why do we use `break` and what will happen if we don't? What is the *fall-through* effect? What is the `default` case and why should we have one? Can an `if` statement always be converted into a `switch`?

- **Loops**
  `for`*,* `while` *and* `do-while`: syntax, use. Pre-test and post-test loops. What are the effects of `continue`, `break` and `return` in a loop? Why are floating point control variables dangerous?

- **Arrays**
  Basics: What is an array? How is it declared? What does its name represent? How do you pass an array as an argument to a function? How do you pass an array element as an argument to a function? Can a function return an array? What types of elements can an array have? Don't forget about zero-indexing.

  MultiD arrays: How do you declare and initialize a two-dimensional array? How do you pass it to a function?

- **Enumerations**
  Basics: What is an enumeration? How is it defined and used? How do we loop through the values of an enumeration? Why do we use enumerations? How is an enumeration variable represented internally?

  Note: structs will not be on the midterm (CONTINUED ON NEXT PAGE)

- **Pointers**
  Basics: What is a pointer variable? How do you declare and initialize one? Why are pointers dangerous? How are pointers related to arrays? Operators `*, &, ->`. Pointer arithmetic.

  Memory issues: What is NULL? What is dynamic memory allocation? `new` and `delete`. Using pointers to implement dynamic arrays.

  Functions: How do we pass a pointer to a function? Why should we be careful with functions returning pointers?

- **C-strings**
  Basics: What is a C-style string? What does *null-terminated* mean? How do you declare and initialize a C-string?

  The `cstring` *library*: You should know and be able to use the following functions: `strcpy, strncpy, strcat, strncat, strcmp, strncmp, strstr, strchr, strlen`

- **Classes**
  Basics: What is the idea behind OOP? What is a class and what is an object? How is a class defined? What are `private` and `public` members? What is *information hiding* and how is it enforced? Why and how do we separate the interface from the implementation? How is conditional compilation achieved?

  Special functions: What are the constructors, destructor, copy constructor? When and how are they called? When and why are they necessary? What is overloading? What operators do we usually overload and how? What are the main issues when overloading the assignment operator? What are `friend` functions? Why and how do we declare/define/use them?