**Concepts From Exams 1 and 2**

* program components, variables, data types and how to choose them, various arithmetic operations, ways to control formatting of output, if-then-else statements, relational and logical operators, loops,

**Chapter 6**

* **functions:** modularization, reuse, return type, parameter list, special void return type
* **function prototypes:** required in real code, and required by clang-llvm compiler
* **Javadoc:** the standard way to document the purpose, parameters, and return value of a function
* **parameters:** formal parameters vs. actual parameters (aka parameters vs. arguments), formal: type and name, actual: value (could be variable, expression, literal, etc. . .)
* **naming functions:** functions *do* something so names should contain verbs
* **return statements:** where they can appear, make sure functions always return, never return from the middle of a loop
* **global variables:** never acceptable, global *constants* are acceptable if used in multiple functions
* **local variables:**defined inside functions, scope is function body, parameters are used as pre-initialized local variables
* **pass-by-value parameters:**argument is *copied* into formal parameter when function is called, formal parameter can be used as a variable, but it is a copy of the original
* **pass-by-reference parameters:**reference variable is a reference to another variable, declared using ampersand (&), an *alias* for another variable, changes actually change real variable, arguments must be variables
* **function design:** a function should do only one thing

**Chapter 7**

* **arrays:** various ways to declare and initialize arrays, distinguishing type of index (always unsigned, use size\_t) versus type of data (no restriction), access elements by position, size fixed at compile time, importance of bounds checking
* **range-based for loop:** aka the foreach loop, loops through array of values automatically, use a reference variable to modify elements of the array, usually used with auto instead of explicit type
* **whole array assignment & comparison:** only way is item by item, usually using a loop
* **common array algorithms:**print contents, sum contents, compute average, find min or max, find position of min or max or some arbitrary element
* **parallel arrays:**same position used for data about the same entity, but stored in different arrays, perhaps of different types
* **arrays as function parameters:** always pass-by-reference, need to pass the size separately, should be const if array is unchanged in function
* **multi-dimensional arrays:** relevant to lots of real-world situations, double-subscripting for two-dimensional arrays, nested for loops are especially helpful, you need to have a mental image of the data to know how to use the array, use dimension parameter names to help with this, passing multi-dimensional arrays as parameters, must specify all dimensions except the leftmost
* **array problems:** static size, size must be known at compile time, arrays don’t know their size, lack of bounds checking
* **vectors:** from the Standard Template Library (STL), how to declare, how to initialize, how to add values (.push\_back), use of size\_t, .size() to determine size, .at(index) to access elements, always pass by reference or const reference

**Chapter 8**

* **searching:** linear search is pretty much the only option for unsorted data, usually implemented with a while loop, main operation is comparison, return position of matched element or size (as opposed to –1) to indicate item is not found, *analysis:* requires *n*/2 comparisons on average when item is found, *n* comparisons when item is not found
* **enhanced linear search:** can be used if the elements are in order, but in that case, much better to use:
* **binary search:** useful for searching sorted lists, know the algorithm, be able to specify what elements are examined during a binary search, be familiar with implementation, *analysis:* takes log2 *n* comparisons, cuts search space in half at each step, know powers of 2
* **sorting:** put values in an array or vector in nondecreasing order, understand bubble sort and selection sort algorithms and be able to work through them yourself, understand how code works