130 Course Content Summary (Fall 2024)

This summary is meant to highlight what type of material is and is not testable. Many questions may rely on you understanding and applying this knowledge; it is not sufficient to memorize this list, you must be able to use the material.

1. General thoughts

- You will have to write and read code on the test.
- You will not be asked anything on VS Code, or GCC/g++ command line.

2. Lecture content

Before Midterm

0. Administrative & Academic Honesty

- Review expectations and consequences for academic honesty.
  Be warned: I am **passionate** about it!
- Nothing testable.

1. Computers Introduction

- General sense of intelligence of a computer vs a human brain.
- Understand “What is computer Science” (don't memorize).
- Know hardware vs software.
- Understand Euclid's algorithm.
- What is an algorithm, a data structure (enough to relate to a program which plays a game); what are the 4 steps in developing a program.
- Language Hierarchy: Machine language, assembly, high-level languages
- Understand C++'s advantages
  - Don't need to memorize C++'s history

2. Introduction to C++

- Correct structure for a simple program (main() method, return type).
- Case sensitive,
- Build process: role of a compiler and an IDE vs command line process (don't need to memorize commands).
- `cout`:
  - “`#include <iostream>`” and “`using namespace std;`”
  - stream insertion operator, line feeds (2 ways), and special characters.
- Error types: Compile, Run-time, Logical.
- Debugging and QA.

3. Variables

- Identifier naming rules
- Variable declaration, initialization, use, naming convention (camelCase)
  - Importance of variable naming.
- Know operators: +, -, *, /, %, =.
  - Know integer division.
- Able to solve problems using C++.
- Know data types; `string`, `char`, `int`, `double`
- Allocating a minimum column width for a conversion specifier with `setw()`
- `cin` to read an integer or a string.
- Scope, uninitialized variables.
- Commenting styles.

4. Expressions

- Expressions & Operators
  - Arithmetic operators, integer division, brackets.
  - Know operator precedence table; summary of currently seen operators is below.

Operator	Description	Associativity
() [] ++ --	Functions, arrays, postfix inc/decrement	Left to Right
! + - ++ --	Not Unary sign operators Prefix increment	**Right to Left**
(type name)	Casting	**Right to Left**
* / %	Multiply, divide, modulus	Left to Right
+ -	Add, subtract	Left to Right
< <= > >=	Comparison	Left to Right
== !=	Equality, not-equal	Left to Right
&&	And	Left to Right
||	Or	Left to Right
= += -= *= /= %=	Assignment	**Right to Left**

  - Expression trees: construction, and use for evaluating expressions.
  - -=, +-, *=, /=, %=
- Constant declaration, use, and naming convention
  Must be able to create and use `const` variables and explain their purpose
- Overflow / underflow; `INT_MAX`, `INT_MIN` - <climits>

5. If & Loops

- boolean expressions: ==, !=, <, <=, >, >=
  - Understand how the following interrelate: true, false, 1, 0, and non-zero values.
  - Find common error of = vs ==
- If statement:
  - if, else, nested if-else,
  - Scope of variables.
  - Good style: indentation, use of {}.
- While Loop
  - x++, x--
  - Reading and using while loops, recognizing infinite loops, use of nested loops.
  - Handling user input in a loop.
- General:
  - Able to trace through a loop, track values, determine outputs.

6. Functions

- Why use functions?
  - Header, return type,
  - Definition vs use vs prototype.
- Arguments
  - Able to write functions that accept parameters, and able to use (call) with arguments.
  - Able to trace through a program.
  - Terminology: arguments, parameter list, parameters
- Returning a value.
  - Know the difference between returning a value and outputting a value.
  - How to call a function and use a returned value
  - Understand execution flow through functions which use return.
- Scope: local, global
  - Scope and lifetime for local, and global variables.
  - Able to explain what is good or bad about global variables and global constants.

7. Random, AND/OR

- Random
  - rand(), srand() - <cstdlib>
  - time(nullptr) - <ctime>
  - How to generate a random number between 1 and 50 (for example).
  - How seed numbers affect the pseudorandom number generator.
- Complex boolean expressions: &&, ||, !
  - Know truth-tables
  - Know precedence table
- Quick test for boolean
- Explanatory variables

8. Functions (again)

- Pass by value
- How and when to use a function prototype.
  - Where to put ;'s
- Understand effect of the compiler flags: `-Wall`, `-Werror`

9. For loops

- Understand break and continue.
- for
  - Definite vs indefinite loops.
  - Syntax of a for loop
  - Variables declare in for loop's initialization exist only in the loop.
  - Able to convert between a while loop and a for loop.
  - Nested for loops

Other skills

- Formatted output (`fixed`, `setprecision()`)
- Debugger
  - Know what the debugger is.
  - Understand the steps of setting a breakpoint, run, step-over.
- Writing clean code (formatting, indentation, naming, “paragraphs of code”)

After Midterm

10. Representation

- Data Types:
  - Bits, bytes, larger sizes (kB, MB, GB, TB)
  - Understand different options for integers (don't memorize).
  - `sizeof()`
- Binary Representation
  - Able to count in base 2, 10 and 16; how to indicate a value is hex.
  - Convert unsigned numbers between base 2, 10, 16.
  - Able to add binary numbers, know about overflow.
  - Negative numbers:
    - Understand what a complement is; no need to compute base 10 complement.
    - Able to convert positive/negative numbers between base 10 and 2's complement
    - Able to do subtraction via addition with complement numbers.
    - Know how to interpret binary values as signed or unsigned.
  - Able to:
    Express a positive/negative integer in 2's complement notation.
    Apply the 2's complement to a positive value.

11. Data Types

- Floating Point:
  - Know float, double; understand long double.
  - Understand issue with exact values and floating point numbers.
- Conversions:
  - Know truncation and rounding with `ceil()`, `floor()`, and `round()`
  - Know what to be a type promotion and a type demotion.
  - Understand rank hierarchy of types; know relative positioning for `char`, `int`, `float`.
  - Implicit conversions: Know rules, when they apply, what they do.
  - Explicit conversions: Know how and when to use it, and what it does.
- Math `<cmath>`
  - `abs()`, `sqrt()`, `pow()`, `ceil()`

12. Stack Memory

- Know the basics of computer memory and addressable bytes
- Know how the stack operates
  - What is push, and pop
  - During a function call, know when values are pushed, and popped
  - How arguments are passed to a function for pass by value
  - Know stack frame, local variables on stack
  - How a return value is returned from a function
  - Able to draw a stack for a simple function call, as shown in class
  - Understand how memory is reused between function calls

13. Vectors

- What they are, when to use them.
- How to create and use them.
- Initializing, adding elements, accessing elements, getting the number of elements.
- Know the difference between a vector element and its index.
- Out of bounds errors, type of error it triggers.
- How to pass a vector to a function as an argument: able to write the function, and call the function.
  - Know if changes to a vector argument inside the function apply to the original.
  - Know pass-by-reference

14. Strings and For-Each loop

- Strings
  - Know how to use a string object (create, concatenate, get size, access/change specific characters).
  - Able to `cin` and `cout` strings.
  - Able to write functions to compare strings and do character-by-character algorithms on them.
  - Know pass-by-constant-reference.
- For-Each loop
  - Know what the for-each loop is and how to use it on a vector and string.
  - Know benefit and limitation of for-each loop.
  - Able to read/write code using a for-each loop. Able to convert code between the for-each loop and the standard for-loop and back.

15. Files

- Know about volatile and non-volatile storage.
- Know what a stream is, and the operators required to work with input vs output streams.
- Know what a **class** is, and what an **object** is. Know what **instantiation** is. Know the dot operator.
- Know how to open, read from, and close an input file.
- Know how to open, write to, and close an output file.
- Know what it means for data file to be white-space separated.
- Know how to read a file line-by-line.
- Know `exit()` vs `return` vs `break`

16. Structures

- Know what parallel vectors are, and their limitations.
- Know what is a structure, how to declare one, why they are useful.
- Know how to use and change an attribute's value in a structure (ex: `myStruct.height = 1;`)
- Understand how to initialize a structure.
- Know how to pass a structure to a function using pass-by-value and pass-by-reference. Know how to return a `struct` from a function.
- Know how to create a vector of structures.

17. Pointers

- Know how to use pointers:
  - What they are
  - How to declare them, how to initialize them (`nullptr`).
  - How to get an address of a variable.
  - Dereferencing a pointer.
- Pass by: value, reference, and pointer
  - Able to trace a program using any of these.
  - Able to read and write a simple function using pointers

18. Arrays & Dynamic Memory

- Arrays:
  - Know how to create an array, access elements. Know that its size is fixed once created.
  - Know how to pass an array to a function.
  - Understand arrays and pointers and their interchangeability.
    - Able to access an array using pointer syntax; able to access a pointer using array syntax.
- Dynamic Memory
  - Know code storage, static memory, automatic memory (stack), dynamic memory (heap)
    - Know what goes in each of these memory areas.
  - Know problem of returning a pointer to a local variable.
  - Understand a dynamic array with `new` and `delete[]`.
  - Know where pointers (as local variables) are stored.

19. Searching

- Understand problem & terminology: target element, search pool.
- Linear search:
  - Understand idea.
  - Able to perform a linear search and count number of elements compared.
  - Able to write a linear search.
  - Able to call a linear search function.
- Binary search:
  - Understand idea, and limitations of when algorithms applicable.
  - Able to apply the binary search algorithm to a data set and count number of elements compared.
    Note: You should be able to do this without being given the algorithm
  - Understand binary search algorithm (you will not be asked to write the algorithm).
- Understand factors affecting which search algorithm you would choose.

20. Sorting

- Understand sorting problem.
- Selection sort & Insertion sort
  - Understand idea of each algorithm.
  - Able to apply each algorithm to a data set showing each pass of the algorithm and what has been sorted.
  - Understand each algorithm's implementation
  - Able to write code to call a sort algorithm, able to trace a sort algorithm if provided.
- Understand criteria for algorithm selection.

21. Recursion

- Understand recursive thinking: base case & recursive step.
  - Able to briefly describe the difference between recursion and iteration.
  - Understand that each recursive call gets its own stack frame.
- Able to write or trace through simple recursive methods.
  - Understand recursive `sum()`, factorial, Fibonacci, odd/even
  - Operations on a vector or array:
    count occurrences of a character, sum values in a vector/array.

