EECS 281 Supplemental Class Notes ("IA Notes")

In Development, 2018-

Hello everyone! The following page hosts a collection of class notes that were written for the class, covering all of the material in the course.

I began this project in the summer of 2018 as a way to convert the material into an accessible text format, as well as to consolidate all of the information for this class into one location. This resource was definitely something that was requested by a lot of students during my time as an IA, especially since EECS 281 is one of the most important classes in the computer science program.

The first 18 chapters of these notes were initially released on the Fall 2020 Piazza forum, and the remaining chapters will be completed sometime in the near future. *The most updated version of the notes will always be available on this page!*

The great thing about EECS 281 is that there is always something new to learn, no matter how much experience you have! Even during my last semester as an IA, I've learned things that I never knew when I first took the class. Hopefully, these notes will make it easier to learn all the concepts in this class, especially as the class size continues to grow each semester. EECS 281 is a challenging class, but it is also a fulfulling one! I wish you the best of luck in this class, and I hope you find these notes helpful.

If you find any errors or want to make any suggestions, please fill out this form or post in the **#class-notes** Discord channel! Your feedback is greatly appreciated.

These notes do include some bonus material that will NOT be directly covered in this class (there's only so much that the class can cover, so this additional content is designed for self-study). This bonus material may include helpful content for future classes or job interviews, or general C++ knowledge that is good to know. Text in this color indicates bonus content that can be skipped if you only want to focus on material you are responsible for knowing.

In these notes, optional material is marked with an asterisk.

The combined PDF document that contains all the chapters (with bookmarks to support easy navigation) can be found at this link (last updated July 19, 2023).

Update September 3, 2023: I've uploaded a few draft practice questions for chapters 1-11. These questions have not been proofread or edited yet, but since the school year is starting and I won't be able to work on this for a few months, I decided it was better to release everything I have now. As a result, there are most likely issues in this problem set that I have not been able to catch, so if you see anything wrong, please let me know! I probably won't be able to make any changes until October at the earliest, however.

Since I am no longer on staff anymore, consider these questions as unofficial. Always prioritize resources from the current staff, which may better reflect the content and difficulty of questions you will see on assignments and exams. This also means that the staff may not be familiar with any of the questions provided here, so if you think something needs to be fixed or improved, make the suggestion in a place that I can access (see instructions above on how this can be done).

Current progress:

- Finish writing content:
  [▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩] (27/27 complete)
- Edit and finalize content:
  [▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩] (27/27 complete)
- Add practice problems/exercises per chapter:
  [▩▩▩▩▩▩▩▩▩▩▩▩▩░░░░░░░░░░░░░░] (13/27 complete)

All errors that were reported to the form before July 19, 2023 have been fixed. Last updated November 19, 2023.

Chapter 1. Programming Foundations [pdf] [practice (DRAFT)]

- This chapter is primarily review for 280 concepts. Most of these concepts are provided to help you on projects.
- Section 1.3 covers member variable organization, which is primarily covered in EECS 370. However, knowledge of how to optimally organize member variables in a class or struct can be used to reduce your memory usage for assignments (which can affect your score). You do NOT need to memorize all the types and their sizes and value ranges (although they are good to know).
- Section 1.7 covers the lifetime of variables and how namespaces work. This is good information to know, but isn't something you'll have to directly work with in this class.
- Section 1.8 deals with different keywords in C++. The `mutable` and `explicit` keywords are covered later in the class, and you may see them on project 2. The `const` and `static` keywords should have been covered in EECS 280, so this should be review.
- Sections 1.9 to 1.17 cover material that may be helpful to understand for projects. However, you can still pass this class without knowing how to use templates or polymorphism (in case you forgot them after taking 280, which is perfectly okay). It's good material to refresh your memory on, since they will occasionally show up in starter files, but you won't be forced to implement something templated or use polymorphism on your own. Similarly, you won't be required to throw exceptions for any of your assignments (although it could make your code cleaner if you did).

Chapter 2. File and Stream I/O [pdf] [practice (DRAFT)]

- This chapter covers streams in C++. Similar to chapter 1, this material is provided to help you on projects, since you will have to read in data from files.
- **The only material that you are required to know well are sections 2.1 to 2.4.** Sections 2.5 and 2.6 cover optimizations that you can take advantage of to speed up I/O, but they go over a bit more detail than needed in case you are interested in why they work. If you are not interested, the important takeaways from these two sections are:
    - Section 2.5: When printing to `std::cout`, it is faster to use `'\n'` to print a newline instead of `std::endl`.
    - Section 2.6: As long as you aren't mixing C++ and C I/O in your program, adding `std::ios_base::sync_with_stdio(false)` on the first line of `main()` can significantly speed up I/O.
- Sections 2.7 to 2.9 cover additional concepts relating to streams (such as stringstreams, file streams, and stream polymorphism) that are useful to know, but you won't be required to use any of them for this class.

Chapter 3. Command Line Parsing [pdf] [practice (DRAFT)]

- This chapter covers command line arguments in C++. Similar to chapters 1 and 2, this material is provided to help you on projects, since you will have to identify and process arguments from the command line.
- Section 3.1 covers `argc` and `argv`, which should be 280 review.
- Section 3.2 covers switch statements, which are used by `getopt_long()`. You will likely see switch statements several times in this class. The concept of enumerated types is also introduced in this section, but enums are covered after the midterm (so don't worry if you don't hear about them before then).
- Section 3.3 covers `getopt_long()`, which you will be using to parse command line arguments in this class.
- Section 3.4 is a newly added bonus section covering a resouce you might see in industry, and not something you are responsible for in the class.

Chapter 4. Complexity Analysis [pdf] [practice (DRAFT)]

- This chapter covers the concept of time complexity and how to measure the efficiency of an algorithm. This chapter forms the foundation of all future content in this class (and is one of the core concepts of 281). You should know this material well!
- For section 4.8, you do NOT need to know any calculus for the class! This is just to introduce another method that can be used to find the complexity class of a function.

Chapter 5. Recursion and Recurrence Relations [pdf] [practice (DRAFT)]

- This chapter covers recursion and methods that can be used to calculate the time complexity of recursive algorithms. You do NOT need to memorize any summation formulas for this class! The math in this chapter is provided for completeness.
- The end of section 5.7 covers intuition behind why the Master Theorem works. You won't be required to know the full details, but this does make understanding the Master Theorem easier. If you just want to memorize the formula, that's perfectly fine for this class.
- The bonus section at the end of 5.7 introduces the extension of the Master Theorem for polylogarithmic functions - you do NOT need to memorize these special cases for exams. Just knowing the three conditions at the beginning of section 5.7 is enough.

Chapter 6. Arrays and Containers [pdf] [practice (DRAFT)]

- This chapter covers memory management and begins a series of chapters on different types of containers, starting with the array.
- For section 6.2, you do NOT need to know how to use the `std::array<>` for this class (although you may have to use it in upper-level classes).
- For section 6.6, you do NOT need to understand how `memcpy()` works for this class. However, you will be seeing this function quite a lot if you continue into upper-level classes that use C++, so it doesn't hurt to familiarize yourself with this function.
- The entirety of 6.9 covers material that is not taught in this class (as of now). However, this is core knowledge for C++, so you are encouraged to look over this material after you finish the class! Move semantics aren't banned on the autograder, so you can also take advantage of them in your assignments if you know how they work, but you will not be required to know how to use them to do well on class assignments.

Chapter 7. Vectors [pdf] [practice (DRAFT)]

- This chapter covers dynamic arrays and the STL `std::vector<>` vector container.
- Section 7.5 provides information on how to optimize the dimensions of a multidimensional vector to reduce memory usage. This is mainly provided to help you on projects. All of the other material should be covered in class (which you should be responsible for knowing).

Chapter 8. Linked Lists [pdf] [practice (DRAFT)]

- This chapter covers linked lists (which should be 280 review). However, lists will still be covered in lab, and you are still required to know them well for exams.
- You do NOT need to know how to use the `std::list<>` or `std::forward_list<>` containers (section 8.6).

Chapter 9. Stacks and Queues [pdf] [practice (DRAFT)]

- This chapter covers stacks, queues, and deques, and how they are used.
- You do NOT need to know the implementation details of `std::deque<>` in the middle of section 9.6.

Chapter 10. Priority Queues and Heaps [pdf] [practice (DRAFT)]

- This chapter covers different types of heaps, how they are implemented, and the STL `std::priority_queue<>` container.
- You do NOT need to know how to use `std::make_heap()`, `std::push_heap()`, or `std::pop_heap()` which is briefly covered in section 10.4.2. However, it doesn't hurt to know that this function exists in case you need to heapify something in a future class. You should NOT use this function for project 2, however.

Chapter 11. Iterators and the STL [pdf] [practice (DRAFT)]

- This chapter covers iterators and different features of the C++ standard library.
- You do NOT need to know how to use `std::tuple<>` in this class (section 11.3). The `std::pair<>` (section 11.2) should be enough, but you can still go through the class without ever using a pair (you can use a struct with two values instead).
- Section 11.5 covers different types of iterator adaptors. Only the reverse iterator is required knowledge for this class. Feel free to read about the stream and insert iterators, but you won't be required to know them for this class.
- Section 11.6 covers auxiliary iterator functions, which are useful to know, but NOT necessary for this class.
- You won't need to memorize all the functions in sections 11.8 and 11.9 (however, knowing they exist can help you a lot with your code). For section 11.9, the only functions you should know well are the min and max functions, `std::sort()`, the variants of `std::find()`, the variants of `std::remove()`, and the lower and upper bound functions.
- Sections 11.10-11.13 cover optional material that you aren't required to know for the course. However, learning how to use lambdas is a useful skill, and learning how to use random number generators can be valuable for generating test files (especially for upper-levels that require extensive testing). The section on C++17 features is also good to know, even if you aren't yet able to use them.

Chapter 12. Amortization and Amortized Analysis [pdf] [practice (DRAFT)]

- This chapter covers amortized analysis, which can be used to establish a tighter complexity bound on a sequence of operations.

- The only method you need to know for this class is aggregate analysis (section 12.2). The other two methods (12.3 and 12.4) are NOT covered in this class, and you aren't required to know them. These methods are included for completeness of the concept.

Chapter 13. Sets and Union-Find [pdf] [practice (DRAFT)]

- This chapter covers sets, disjoint sets, and union-find.
- You will NOT be required to know the optimizations of union-by-size or union-by-rank, covered in section 13.4, although they are useful to know.

Chapter 14. Sorting Algorithms [pdf]

- This chapter covers different sorting algorithms. The content in here should all be covered in class. After 2022 edit: section 14.10 is bonus material.

Chapter 15. Binary Search and Additional Algorithms [pdf]

- This chapter covers binary search and several additional algorithms that are covered (but don't fit in any of the other chapters). This chapter is basically a "catch-all" for midterm content. However, the additional algorithms covered in sections 15.3 to 15.8 aren't as important as the other material covered before the midterm, so don't stress too much over them. For example, you probably won't be asked to implement Moore's voting algorithm or Misra-Gries on an exam or anything.

Chapter 16. Strings and Sequences [pdf]

- This chapter covers strings, sequence operations, and string searching algorithms.
- You do NOT need to memorize all the C string operations covered in 16.1.
- You do NOT need to know the implementation of the GCC string, covered at the beginning of 16.2.
- The KMP string searching algorithm (section 16.7) will NOT be covered in this class. However, along with Rabin-Karp, KMP is one of the more important string searching algorithms out there, and is good to understand even if you will not be required to know it for this class.

Chapter 17. Hash Tables and Collision Resolution [pdf]

- This chapter covers hashing, hash tables, and hash collision resolution, as well as C++ containers that utilize hashing.
- Section 17.7 introduces the unordered multimap and unordered multiset containers. However, you do NOT need to know how to use them. The takeaway from this section is that you can map a key to

multiple values by using a `std::unordered_map<>` that maps each key to another container (such as a `std::vector<>`) instead of using a multimap.
- Section 17.8 introduces the concept of composite hash functions, which is briefly covered in lecture. You should know how to use `std::hash<>`, and you should know that linearly combining hash values is not the best way to hash a custom object with many components. However, knowledge of how to use the better hash combiner implementation (the one with the seed) is NOT needed for the class. In addition, you will NOT need to know how to implement a composite hash function for a custom object, which is covered at the end of section 17.8.

Chapter 18. Trees [pdf]

- This chapter covers different types of trees (binary trees, binary search trees, and AVL trees), as well as C++ containers that rely on trees.
- You do NOT need to know how to count the number of possible binary search trees using Catalan numbers in 18.6 (this is just a bit of trivia).
- Section 18.10 introduces the multimap and multiset containers. However, similar to section 17.7, you do NOT need to know how to use them.
- Section 18.11 covers the concept of tries, which aren't covered in this class, but do show up during programming interviews from time to time.

Chapter 19. Graphs and Elementary Graph Algorithms [pdf]

- This chapter covers graphs, graph ADTs, and elementary graph algorithms.
- Section 19.6 covers topological sort, which isn't explicitly mentioned in class, but could show up during interviews. Also, topological sorting algorithms are just an application of BFS/DFS, so technically they are fair game for exams (that being said, I don't think it's as likely, so I've marked this as optional material to study for interviews, not not necessarily for this class).
- Section 19.7 covers strongly connected components, which you do NOT need to know about, but is an interesting concept nonetheless.
- Section 19.8 covers algorithms for finding articulation points and bridges, which is also something you do NOT need to know for the class.

Chapter 20. Minimum Spanning Trees [pdf]

- This chapter covers minimum spanning trees, Prim's algorithm, and Kruskal's algorithm.
- You don't need to know the details of how a Fibonacci heap works.

Chapter 21. Greedy Algorithms and Divide-and-Conquer [pdf]

- This chapter covers the algorithm families of brute force, greedy, and divide-and-conquer/(combine-and-conquer).
- Section 21.3: This isn't 203, so don't worry too much about all the proof stuff. Just know the characteristics of a greedy algorithm and common problems that can be solved using the greedy approach (particularly the interval selection problem). Also, don't worry about Huffman Coding - I included it since it was previously covered in Maxim Aleksa's 281 notes.
- Section 21.4: I've included both the formal definition of divide-and-conquer used by most algorithms textbooks and future algorithms classes (like 376), as well as the definition used in 281 (splitting the dividing and combining steps with a new family called "combine-and-conquer"). If you are ever given an exam question or anything on this where the answer depends on the definition, assume it uses the 281 definition.
- Section 21.4: Don't worry too much about Kadane's algorithm.

Chapter 22. Backtracking and Branch and Bound [pdf]

- This chapter covers backtracking and branch and bound, as well as the traveling salesperson problem (TSP).

Chapter 23. Dynamic Programming [pdf]

- This chapter covers dynamic programming, and introduces examples of patterns that can be used to solve common dynamic programming problems.
- This chapter is pretty heavy on examples - if you don't have time, I would focus on the following problems: Fibonacci, binomial coefficient, counting ways in a grid, stairs, knights on a chessboard, minimum sum path, coin change, house robber, subset sum, pipe welding, longest common subsequence (returning length only), and longest increasing subsequence. Also, dynamic programming is best learned through experience and practice.
- As always, you don't need to worry about any terms that were not covered in class.

Chapter 24. The Knapsack Problem [pdf]

- This chapter covers variations of the knapsack problem and algorithms that can be used to solve them.
- Sections 24.4-24.5: Don't worry too much about the bounded/unbounded knapsack variants (but feel free to give them a read if you are curious, since you have learned all the material relevant to these two sections, and there really isn't any new material).

Chapter 25. Shortest Path Algorithms [pdf]

- This chapter covers shortest path algorithms that can be used to find the lowest-weight cost between vertices of a weighted graph.
- The only material that you will need to know for this class is 25.1 and 25.2 (Dijkstra), excluding the proof of Dijkstra's in section 25.2.2.
- Sections 25.3-25.6: Goes over additional shortest path algorithms that we don't cover due to time. Still useful to know, but not required material for 281.

Chapter 26. Computational Geometry [pdf]

- This chapter covers computational geometry.
- We usually never get this far, so I took a bit more liberty with this chapter and explored some of the topics covered in the slides in a bit more depth (since I'm assuming the people who end up reading this want to explore this topic in more detail).
- Knowing a high-level overview of 26.1, 26.3-26.6, and 26.11 is probably enough for this class, but (from experience) there is a fairly good chance that this topic will not show up on the final exam unless otherwise specified. If a computational geometry question does show up, it probably will not be super involved.
- Sections 26.2, 26.7-26.10: Goes over some other concepts introduced in the slides but aren't covered as much in depth, so these are just here as an additional reference if you want to learn more about the material.

Chapter 27. Smart Pointers and Memory Management [pdf]

- The chapter covers the bonus concepts of smart pointers and memory manangement. This content is NOT part of class material, but it is here as an additional reference since smart pointers are important to know, not just for later classes, but also if you ever work with C++ in industry.