**CS273-1 Syllabus**   
**Data Structures and Algorithm Analysis**   
**Summer 2017**

|  |  |
| --- | --- |
| **Instructor:** | Dr. Peter A. Tucker |
| **Contact Information:** | Office: JOHN 307  Email: [ptucker@whitworth.edu](mailto:ptucker@whitworth.edu)  Phone: (509) 777-4664 |
| **Office Hours:** | I’m available and in my office many times during the week. Please feel free to drop by and visit or to ask for hints! |
| **Course Times and Locations:** | Independent study |
| **Course Description:** | Data structures and algorithms are the heart and soul of computer science. Well designed (and well tested!) **data structures** and **algorithms** form the **building blocks of all popular computer applications** ranging databases to business applications, and from scientific programs to games. Employers expect that computer science students will have a good grasp of this content. The main focus of this course will be to learn how to build the most important (and thus most common) data containers, (e.g. vector, stack, list, tree, map, hash table, etc.) and the most important algorithms (e.g. searching, sorting and graph algorithms).  In close coordination with the text, this course places a strong emphasis on problem solving and software design. The course takes a careful and systematic approach to the description and development of each of the fundamental data structures – encouraging students to “Think, then code”. The course starts by describing software design principles using “Abstract Data Types” (ADTs) and how these relate to an object-oriented description of data structures. Once students are familiar with a formal representation of data structures the course moves on to illustrate how different data structures are designed and analyzed for efficiency, tested for correctness and applied to solve computational problems. |
| **Course Objectives:** | 1. To gain a working knowledge of the most common **data containers** (i.e. the building blocks of algorithms): Lists, Stacks, Queues, Trees, Graphs, Functors (function object, lambda function…), Trees, etc. 2. To gain an introductory understanding of algorithm analysis. 3. To understand how to specify the design of data containers using **Abstract Data Types (ADTs)**. These help us design data structures in such a way that they are independent of a particular machine, language or operating system. 4. To understand how to use an important library of data containers, the **Standard Template Library (STL)** 5. To understand how to **manage memory dynamically** at a low level using **new**, **delete**, and **pointers,** and **smart pointers**. 6. To understand the **properties of computer algorithms** and the importance of designing efficient, safe, testable, and re-usable code in a cost effective manner. 7. To understand and learn **basic software engineering paradigms**: object oriented programming, inheritance, recursive algorithm design. 8. To develop your ability to use **software engineering tools**: unit testing using a software testing suite, source code control software such as Git, using the command line to compile and link code, and using makefiles to automate this process. |
| **Learning Methods:** | Students will complete programming projects, in-class tutorials, and exams that reinforce concepts learned through reading and Session. The course will also include days for review, reflection, and response. Finally, the course will contain two midterm examinations, and a final project / presentation. |
| **Text:** | |  |  | | --- | --- | | http://ecx.images-amazon.com/images/I/51dhwlJG7-L._SX258_BO1,204,203,200_.jpg | **REQUIRED TEXT**  **Objects, Abstraction, Data Structures and Design: Using C++**  Authors: Elliot B. Koffman & Paul A. T. Wolfgang, Temple Univ.  Publisher: Wiley  ISBN: 978-0-471-46755-7 | |
| **Department Goals (DG) Addressed in this Course:** | 1. **Demonstrate an appropriate theoretical foundation in computer science.**  This class provides an introduction to the theory of computer science. We focus on the theory of abstract data types as well as the theory of algorithm run time complexity (run time analysis) 2. **Develop the proficiency for software engineering.**  This class also provides an introduction to the tools and techniques of software engineering. Everything from data design to the tools used in software engineering. 3. **Cultivate problem solving and critical thinking skills.**  In this class we encourage students to think critically and deeply about the basics of computer science. 4. **Understand the need for sound ethical decision-making and consider how faith and/or worldview may inform one’s vocation and professional practices.**  In this class we encourage students to think about how their values, faith and attitudes affect their professional integrity, work ethic and interpersonal dealings. |
| **Attitudes of Successful Students:** | The students who have the greatest success after college have learned to practice the skills and attitudes of critical thinking: [[1]](#footnote-1):   1. **Passion for learning** - Successful students believe they can learn and have a passion for learning and discovery. They don't fall into the trap of taking a class for a "grade". 2. **Seeking to understand** - Successful students want to understand deeply. They break difficult problems into pieces. They seek to find connections between disciplines and ideas. 3. **Evaluate options and approaches** - Successful students use logic and reasoning to compare and predict the outcomes of particular choices. |
| **Requirements:** | **Attendance**   * Attend class meetings regularly and consistently * Please inform the instructor if you need to miss class (see the policy for unexcused absences in the grading standards section of this syllabus).   **Attention and Asking Questions**   * Questions are encouraged. * Ask for clarification whenever you are confused or need further information on a particular topic presented in class. * Do not denigrate (belittle, make fun of) other student’s questions or requests for clarification. A question you may think irrelevant, may actually spark a discussion that generates learning.   **Assignments**   * Complete assignments on the designated due date and in the manner specified in the instructions. * You can expect to spend a minimum of 3 hours outside of class for each class period. * Put all programming projects created for this class in your personal folder on CS1: **\\CS1\CS\_Students\username\CS273**\, etc… * Keep backup copies (somewhere else besides CS1) of all your assignments and projects that you turn in. * Keep all assignments, projects, and exams that we hand back to you. * Write clean legible code that meets the specifications (i.e. works correctly) and contains meaningful comments. * Consult Blackboard (<http://go.whitworth.edu> ) to find the most current class information etc. |
| **Academic Honesty Policy:** | Please note that I take extremely seriously the university’s policy on the need for academic honesty in all your work. **I refer you to the Whitworth Catalog, and the current Student Handbook, where guidelines on plagiarism and other forms of academic dishonesty are spelled out. Any form of dishonesty in an assignment will lead to a zero on the assignment, and I reserve the right to give a grade of F for the course as well.** The following examples are not an exhaustive list. These are simply provided for clarification:  **Examples of academic honesty for this class:**   * Turning in one’s own work * Obtaining help from either the instructor or course grader * Discussing assignments with fellow student(s) * Submitting one copy for multiple students for projects where the instructor specifically permits or requires students to work together * Getting or giving help with syntax errors, runtime errors or on how to run the computers   **Examples of academic dishonesty (cheating) for this class:**   * Turning in other’s programs or homework as your own * Copying other’s programs, tests, quizzes, homework answers, etc. * Submitting one copy for multiple students on individual assignments * Copying someone else’s work and then modifying it to look like your own work |
| **Grading Standards:** | “A” students will take the initiative to seek out ways extend their learning and understanding of the course concepts.   |  |  | | --- | --- | | **Course Component** | **% of Final Grade** | | Assignments | 40% | | Exam 1: Written & Practical | 20% | | Exam 2: Written & Practical | 20% | | Final Exam: Project & Writeup | 20% |  |  |  | | --- | --- | | **Total % Earned** | **Grade** | | 90% - 100% | A- to A | | 80% - 89% | B- to B+ | | 70% - 79% | C- to C+ | | 50% - 69% | D- to D+ | | 0% - 49% | F |   **Policy on Late Work:**   * You must make a serious effort to turn assignments in on time. We understand there may be illness or other factors that affect your ability to turn some assignments in on time. *Contact both your instructor and the grader for the class by email one day in advance of the due date if you know it will be late*. * Otherwise assignments submitted late without prior written (email) instructor approval - will have 10% deducted the first day late, 20% deducted the second day late, and 50% deducted for three to five school days late. * No assignment or projects will be accepted more than five school days late (if you have extenuating circumstances, talk to the instructor ahead of time).   **Policy on Unexcused Absences:**  Students with either **three** **consecutive** unexcused absences or **eight** **total** unexcused absences will automatically receive a grade of WF at the end of the semester. |
| **Evaluation Guide for Programming Problems:** | NOTE: The following general guide illustrates in general how your programming assignments are assessed. Individual assignments may vary.   |  |  | | --- | --- | | 25% | **Problem analysis**: Does the design exhibit a **general understanding of the problem**, even though it may not work correctly? | | 50% | **Execution and Results**: Does the project **work correctly** and generate the correct answer(s) according to the problem specifications? | | 15% | **Quality and Maintainability**: Is the project **well designed**, does it have appropriate comments and contains clean code that follows recommendations given in class, the book and study guides? | | 10% | **Efficiency and Creativity:**  Top projects designs show **ingenuity**, **creativity**, and **efficiency** as well as being **readable and easy to follow**. | |
| **Special Needs:** | Whitworth University is committed to providing its students access to education. If you have a documented special need that affects your learning or performance on exams or papers, you will need to contact the Educational Support Office (Andrew Pyrc – ext. 4534) to identify accommodations that are appropriate. |
| **MSDN Academic Alliance:** | Whitworth University joined the Microsoft Developer Network Academic Alliance (MSDNAA). Through this alliance students are able to download many of the Microsoft software products developers use at no charge, including Visual Studio .NET.   You will receive an email early in September from e-Academy with instructions on how to get those products. **Be sure and print out this email or save it in a safe location on your computer so you can access these Microsoft products for free!** |

**CS273-1 Tentative Schedule (Summer 2017)**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **TOPIC** | **READING** | **DUE** |
| 23 May | Session 01: Introduction to the Course | P.1 – P.9 |  |
| 24 May | Session 02: Introduction to Software Engineering | 1.1 – 1.3 |  |
| 25 May | Session 03: Abstract Data Types & Case Study | 1.4 – 1.8 | HW1: C++ Review |
| 26 May | Session 04: Testing & Lab: Unit-Testing Framework | 2.1 – 2.2 |  |
| *29 May* | *Memorial Day* |  |  |
| 30 May | Off day |  |  |
| 31 May | Session 05a: Source Code Control & GIT  Session 05b: Program Errors and Exceptions | Sup & 2.3 – 2.5  Sup. Reading | HW2: Soft. Eng. |
| 01 Jun | Session 06: Algorithm Efficiency | 2.6 |  |
| 02 Jun | Session 07: Inheritance & Composition | 3.1 | Unit Testing Lab |
| 05 Jun | Session 08: Polymorphism | 3.2 – 3.4 | HW3: Exceptions |
| 06 Jun | Session 09: Namespaces and Design Patterns | 3.5 |  |
| 07 Jun | **Intro to Containers: Lists, Stacks & Queues in Python** | Sup. Reading |  |
| 08 Jun | **Intro to Containers: Sets & Dictionaries in Python** | Sup. Reading | HW4: Python Tutorial |
| 09 Jun | **Exam 1** |  |  |
| 12 Jun | Session 10: Intro to Vectors | 4.1 |  |
| 13 Jun | Session 11: Vector Implementation in C++ | 4.2 – 4.3 |  |
| 14 Jun | Session 12: Managing Memory: Smart Pointers in C++ | 4.4 | HW5: Banking I |
| 15 Jun | Session 13: Linked & Doubly Linked Lists and Iterators | 4.5 |  |
| 16 Jun | Session 14: STL List and Implementation | 4.6 – 4.7 |  |
| 19 Jun | Session 15: STL Lists, Functors, and Algorithms | 4.7 – 4.10 |  |
| 20 Jun | Session 16: Stacks and Stack Applications | 5.1 – 5.4 | HW6: Banking II |
| 21 Jun | Session 17: Queues and Queue Implementations | 6.1 – 6.3 |  |
| 22 Jun | Session 18: Deques and Deque Implementations | 6.4 |  |
| 23 Jun | Session 19: Airline Counter Simulator | 6.5 | Start Simulation Proj |
| 26 Jun | Session 20: Introduction to Recursion | 7.1 – 7.2 |  |
| 27 Jun | Session 21: Recursive Problem Solving | 7.3 – 7.5 |  |
| 28 Jun | Session 22: Trees and Binary Search Trees | 8.1 – 8.4 | HW7: Simulation |
| 29 Jun | Session 23: Heaps and Heap Applications | 8.5 – 8.6 |  |
| 30 Jun | **Exam 2** |  |  |
| 10 Jul | Session 24: Priority Queues and Sets | 9.1 |  |
| 11 Jul | Session 25: Maps and Multi-maps | 9.2 | HW8: Recursion |
| 12 Jul | Session 26: Hashing and Hash Tables | 9.3 – 9.4 |  |
| 13 Jul | Session 27: Quadratic Sorting Algorithms | 10.1 – 10.5 |  |
| 14 Jul | Session 28: Advanced Sorting Algorithms | 10.6 – 10.9 | HW9: 10 years |
| 17 Jul | Session 29: Self-Balancing Tree | 11.1 – 11.2 |  |
| 18 Jul | Final Project Lab Day |  |  |
| 19 Jul | Final Project Lab Day |  | Simulation: Final UML |
| 20 Jul | Final Project Lab Day |  |  |
| 21 Jul | **Final Project Due** |  | Final Project |

1. Based on the article <http://www.engin.umich.edu/~problemsolving/strategy/crit-n-creat.htm> [↑](#footnote-ref-1)