Learning Outcomes - 2021

**** COMP151**

At the end of the course, students

- understand the fundamental syntax & computer programs
- understand the fundamental control and loop (iteration) structures
- program simple algorithms, such as counting, summing, and finding maximum/minimum
- implement, test, and debug simple recursive functions and procedures
- understand the basic data structures used in programming
- argue effectively about the merits and possible unintended consequences of a computing implementation
- effectively write or present about the impact of computing on society. Students should extrapolate from historic lessons learned from unintended consequences of computing to the current computer solutions.

**** COMP152**

After taking this course, the student should be able to

- Design and implement programs using the Object Oriented Paradigm that require multiple classes and structures
- Develop code that responds to error conditions raised during execution
- Design, code, test, and debug simple event-driven programs that respond to user events
- Design, code, test, and debug programs that have Graphical User Interface
- Use industry standard debugging tools
- Discuss the consequences of software piracy on software developers and the role of relevant enforcement organizations
- Understand the uses and limitations of trademark, copyright and patents for intellectual property protection.
- Work effectively in teams.
- Use automated testing to verify correct program behavior
- Install and use a 3rd party library/API

**** 206 (Introduction to Computer Organization)**

- Design simple combinational and sequential logic circuits using fundamental building blocks, including AND, OR, and NOT operators.
- Know how flip-flops, state machines, and arithmetic units are built from logic gates.
- Understand how numbers and text can be represented in digital form and be able to explain the limitations. Know how to convert signed and unsigned numbers to different number systems, including binary.
- Know and be able to explain how to write a simple assembly program or module. Understand how data movement, arithmetic, control flow, subroutine calls, system calls, and I/O are handled at the assembly and machine code level. Appreciate how high-level constructs translate to machine code.
- Understand how instructions are encoded in machine code.
- Understand the relationship between instruction set architecture, microarchitecture, and system architecture and their roles in the development of the computer.

- Understand the basic computer architectural building blocks: registers, arithmetic logic units, control units, buses, and memory.
- Explain the function and behavior of a simple processor at the logic circuit level
- Be familiar with the history of computer technology, from vacuum tubes to VLSI, and from warehouse-scale computers to personal computers
- Be able to physically construct a digital circuit on a breadboard, preferably, or using CAD software.

**** 250 (Data Structures and Algorithms)**

- After taking this course student should be able to:
- Implement abstract data types in multiple ways recognizing the various strengths and weaknesses of those implementations the data types should include
    - arrays
    - lists
    - stacks
    - queues
    - trees
    - cyclic graphs
    - hash tables
- Contrast different implementations through objective means, such as asymptotic analysis (Big-O) of primitive functions (for example: adding, removing, searching, sorting.)
- Apply the concept of algorithmic efficiency to understand that different implementations may be 'best', depending on the requirements of the task.
- Implement at least one efficient sort
- Apply recursion to solve problems in data structures and know when to choose recursion over iteration
- Identify the appropriate data structures for a given problem

**** COMP340 (Organization of Programming Languages)**

At the end of the course, students will:

- Work with the command line tools needed to use the language of instruction
- Explain the differences between different programming paradigms
- Describe history and evolution of programming languages
- Explain the abstractions (e.g., grammars) of programming languages
- Demonstrate how control structures (e.g., if statements, loops) differ among languages
- Describe the advantages/disadvantages of implicit versus explicit variable declarations
- Explain the advantages/disadvantages of strong versus weak type checking
- Describe how different languages implement encapsulation
- Understand the OO paradigm and how it contrasts to non-OO languages
- Explain how different languages (e.g., Java vs. C++) implement object orientation (single vs. multiple inheritance, polymorphism, etc.)
- Use regular expressions

**** Comp 350 (Operating Systems)**

After taking this course the student should be able to:

- Work with the command line tools needed to use the language of instruction
- Understand the fundamental concepts of operating systems.
- Understand the role that interrupts play in a modern operating system. including context switching, kernel-mode privilege and system calls.
- Understand the concepts of process and thread as implemented in modern operating systems by demonstrating their use.
- Understand process scheduling in a multi-programming environment and implement a process scheduling algorithm.
- Demonstrate the use of OS primitive such as semaphores for process synchronization.
- Understand memory management techniques, including virtual memory in the modern operating systems.
- Demonstrate the potential run-time problems that arise from the concurrent operation of separate tasks.
- Understand file system structure and implement a file system such as FAT.
- Work effectively in teams

**** 390 (Software Engineering)**

Software Process:

- Be familiar with traditional and modern software development processes and articulate their advantages and disadvantages.
- Apply iterative and incremental, and/or agile software development practices in a software project.
- Work effectively in teams by understanding principles and practices of good team management, collaboration, planning, and communication.
- Document software development work during design and prior to delivery and understand the importance of software documentation.

Software Models:

- Apply a wide range of possible software development models in software analysis, design, application to specific cases.
- Demonstrate a practical understanding of the evolvability of user needs, and the challenges of communicating with software customers.
- Demonstrate practical experience with creating use cases and formulate functional requirements from user need.
- Understand different aspects and techniques of software testing
- Using CASE tools:
- Use state-of-the-practice software design and development tools.
- Demonstrate knowledge of different licensing options, and the protections and limitations of various intellectual property laws

**** 430 (Computer Networks)**

After taking this course the student should be able to:

- Understand the basic concepts in computer networking
- Implement low-level network protocol techniques (such as sliding window protocols)
- Implement a model of a routing algorithms
- Apply the knowledge of Mathematics in Computer Networking problems
- Understand the structure and function of TCP/IP network layers.
- Understand the application layer protocols such as SMTP, HTTPS, DNS, FTP/SCP
- Understand physical layer concepts such as wired vs wireless and bandwidth.
- Gain experience in the design and analysis of network protocols
- Develop client-server solutions using socket programming.
- Learn the principles of network security.

**** 435 (Analysis of Algorithms)**

By the end of this course student will be able to:

- Analyze the asymptotic performance of algorithms.
- Write rigorous correctness proofs for algorithms.
- Demonstrate a familiarity with major algorithms and data structures.
- Apply important algorithmic design paradigms and methods of analysis.
- Analyze worst and best case behavior of an algorithm.
- Understand the formal definition of Big-O, Big-Omega and Theta notations.
- Use recurrence relations to determine the time complexity of recursively defined algorithms.
- Use efficient algorithms for sorting and searching.
- Understand advanced data structures such as binary search trees, heaps, hash tables and graphs.
- Learning Algorithm design techniques: divide-and-conquer, dynamic programming, greedy algorithms, amortized analysis, randomization.
- Algorithms for fundamental graph problems: graph traversal, connected components and topological sort.
- Apply algorithmic principles to provide a computing-based solution that meets a given set of requirements such as making decisions based on time-space complexity trade-off of algorithms and the characteristics of input data.

**** 490 (Senior Design and Development)**

- Satisfactorily programs part of an existing large project
- Accurately describes the merits and possible unintended consequences of a computational system, service or proposal
- Identifies disruptive effects of latest computing advancements on individuals, organizations, and society
- Can articulate differences/tradeoffs of multiple designs
- Works together as part of a team.
- Builds a significant project using 3rd party library with little formal instruction on those libraries
- Demonstrates use of state-of-the-practice (or state-of-the-art) development techniques including automated testing for 1 & 6 above
- Can work effectively with data and data stores.