Computer Science I 2023-2024 Catalog

[ARCHIVED CATALOG]

CSCI 101 - Computer Science I

PREREQUISITES: Demonstrated readiness for College-level Calculus and SDEV 120 - Computing Logic
PROGRAM: Computer Science
CREDIT HOURS MIN: 3
LECTURE HOURS MIN: 2
LAB HOURS MIN: 2
DATE OF LAST REVISION: Fall 2022

Introduces the fundamental concepts of procedural programming. Topics include data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging. The course also offers an introduction to the historical and social context of computing and an overview of computer science as a discipline.

MAJOR COURSE LEARNING OBJECTIVES: Upon successful completion of this course the student will be expected to:

1. Demonstrate fundamental programming constructs:
    a. Explain the syntax and semantics of a higher level language
    b. Create a program utilizing
        i. Expressions,
        ii. Assignments,
        iii. I/O,
        iv. Iterative and conditional control structures, and
        v. functions.
    c. Utilize an integrated development environment (IDE) to create, execute, test, and debug programs.
    d. Use the techniques of decomposition to modularize a program.
    e. Apply consistent documentation and program style standards that contribute to the readability and maintainability of software.
2. Analyze and explain the behavior of simple programs utilizing variables, expressions, assignments, I/O, control structures, functions, parameter passing, preconditions, postconditions, and invariants.
3. Explain the role of algorithms in the problem-solving including;
    a. Analyze the best, average, and worst-case behaviors of an algorithm
    b. Analyze the performances of an algorithm with various input sizes.
    c. Estimate time and space complexities for a given algorithm using Big-O notation;
    d. Implement a basic numerical algorithm and apply to a given problem.
    e. Discuss the halting problem and why it has no algorithmic solution.
    f. Investigate factors other than computational efficiency that influence the choice of algorithms.
    g. Compare multiple algorithms for a given problem.
4. Utilize fundamental data types including:
    a. Primitive types (Integers, Real Numbers, Booleans, and Characters), Pointers, Arrays, Records/Structures, Strings, Enumerations
5. Analyze machine level representation of data including:
    a. Bits, bytes, and words
    b. Numeric data representation (Binary, Hexadecimal, BCD, 1's Complement, 2's Complement, and Floating Point format)
    c. Non-numeric data (Characters, Images, Sounds, Video)

d. Illustrate color models and their use in computer graphics.
e. Conversion of numerical data from one format to another
f. Affect of fixed-length number representations on accuracy and precision

6. Describe the components of Computer Architecture
   a. Classify the organization and functional units of the Von Neumann machine.
   b. Illustrate how high-level language patterns map to assembly/machine language, including subroutine calls.
   c. Create simple assembly language program segments.
   d. Describe the basic concepts of interrupts and I/O operations.
   e. Compare and contrast different components of the memory hierarchy and memory technology, such as SRAM, DRAM, virtual memory, and cache.
   f. Explain the impact of memory latency on execution time (Von Neumann Bottleneck)
7. Examine major objectives, functions, features, and concepts of modern operating systems.
   a. Discuss the role and purpose of operating systems
   b. Compare prevailing types of operating systems.
   c. Discuss potential threats to operating systems and appropriate security measures.
   d. Diagram the interaction of an Application Programming Interface (API) with an operating system.
   e. Illustrate how applications use computing resources managed by the operating system.
   f. Explain the need for concurrency within an operating system and common methods to implement concurrency.
   g. Illustrate the principles of memory management including virtual memory, paging, thrashing, and partitioning.
   h. Discuss features of an operating system used to provide protection and security.
   i. Diagram the physical hardware devices and the virtual devices maintained by an operating system.
8. Investigate principles of secure design.
   a. Analyze the tradeoffs associated with designing security into a product.
   b. Implement input validation in applications
   c. Discuss the security implications of relying on open design vs the secrecy of design.
   d. Explain the tradeoffs of developing a program in a typesafe language.
   e. Investigate potential errors detected from both strong-type and weak-type languages.
   f. Investigate potential vulnerabilities in provided programming code.
   g. Create programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow.
   h. Investigate common coding errors that introduce security vulnerabilities, such as buffer overflows, integer errors, and memory leaks.
9. Assess human-computer interaction and design issues:
   a. Analyze the importance of human-centered software.
   b. Implement a simple usability test for an existing software application.
10. Discuss software development methodology including:
    a. Fundamental design concepts and principles
    b. Structured and Iterative design
    c. UML (Unified Modeling Language)
    d. Coding Standards
    e. Test-case design (Unit, Integration, System, and Acceptance Testing)
    f. Programming languages
    g. Development environments
11. Apply a variety of strategies to the testing and debugging of simple programs:
    a. Utilize documentation and program style standards that contribute to the readability and maintainability of software.
    b. Conduct code reviews (focused on common coding errors) on program components.
    c. Differentiate between program validation and verification.
    d. Test code for vulnerabilities.
    e. Use software tools to evaluate, test, and debug programs.
    f. Analyze the extent to which another programmer's code meets documentation and programming style standards.
    g. Implement refactoring within given program components.

COURSE CONTENT: Topical areas of study include -

- Abstraction mechanisms
- Human-computer interaction
- Assembly level machine language
- Language translation
- BCD/1's Complement/2's Complement
- Memory Hierarchy
- Binary/Hexadecimal
- Operating systems
- Coding Standards
- Security
- Compilation
- Software design
- Declarations and types
- UML
- Development Environments
- Verification/Validation
- Debugging/Testing

Course Addendum - Syllabus (Click to expand)