Computer Science II

2023-2024 Catalog

[ARCHIVED CATALOG]

CSCI 201 - Computer Science II

PREREQUISITES: CSCI 101 - Computer Science I or (SDEV 140 and CSCI 179)
PROGRAM: Computer Science
CREDIT HOURS MIN: 3
LECTURE HOURS MIN: 2
LAB HOURS MIN: 2
DATE OF LAST REVISION: Fall 2020

Provides a working understanding of the fundamentals of procedural and object-oriented program development using structured, modular concepts and modern object-oriented programming languages. Reviews control structures, functions, data types, variables, arrays, and data file access methods. The course is a second level computer science course introducing object oriented computer programming, using a language such as Java or C. Object-oriented concepts studied include classes, objects, inheritance, polymorphism, operator overloading, exception handling, recursion, abstract data types, streams and file I/O. Students will explore programming concepts such as software reuse, data abstraction and event-driven programming.

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

1. Demonstrate the basic procedural concepts of computer programming through development of programs that utilize:
    a. arithmetic operators, expressions and statements,
    b. reference data types,
    c. input/output,
    d. primitive and abstract data types,
    e. selection and repetition statements,
    f. user defined methods and functions,
    g. collections including arrays, lists, vectors, stacks, and queues,
    h. streams to create and access data files, and
    i. common/standard language libraries.
2. Demonstrate the basic object-oriented concepts of computer programming
    a. Compare and contrast functional and object-oriented programming paradigms
    b. Implement OOP constructs, including encapsulation, abstraction, inheritance, and polymorphism
    c. Utilize immutable and mutable variables in class objects
    d. Use access and visibility modifiers to secure class data and methods.
    e. diagram control flow in a program using dynamic dispatch
    f. Design and implement a simple class hierarchy using superclasses, subclasses, and abstract classes.
    g. Overload functions and operators
    h. Design and implement generic classes with templates
    i. Identify the data components and behaviors of multiple abstract data types.
    j. Apply a variety of strategies to the testing and debugging of programs.
3. Develop Graphical User Interfaces/Event Driven Programs:
    a. Explain basic principles of computer graphics including 2D and 3D objects, transformations, clipping, windowing, rendering, lighting and ray tracing.
    b. Illustrate color models and their use in computer graphics.
    c. Write a simple application that uses a modern graphical user interface.
    d. Create an interactive program using an event-driven style.

e. Discuss the usage of information hiding through steganography in images, messages, videos, or other media files.

4. Discuss software engineering, software maintenance and software reuse:
   a. Illustrate the concepts of modeling and abstraction with respect to problem solving.
   b. Diagram the phases of the secure software development lifecycle (SecSDLC).
   c. Describe the concept of finite state machines
   d. Describe security as a continuous process of tradeoffs, balancing between protection mechanisms and availability.
   e. Illustrate the security implications of relying on open design vs the secrecy of design.
   f. Apply consistent documentation and program style standards
5. Demonstrate basic concepts of networking and data communications
   a. Diagram the basic structure of the Internet.
   b. Diagram the layers of the OSI model, including associated protocols (TCP/UDP, Socket APIs, and Application Layer Protocols)
   c. Categorize the principles used for naming schemes and resource location. NC
   d. Implement a simple distributed network application.
   e. Describe security concerns in designing applications for use over wired and wireless networks.
6. Demonstrate the principles of secure programming and design
   a. Investigate potential vulnerabilities in provided programming code.
   b. Create programs which use defensive programming techniques, including input validation, type checking, exception handling and protection against buffer overflow.
   c. Analyze the interaction between a security mechanism and its usability.
   d. Investigate potential vulnerabilities in provided programming code.
   e. Investigate common coding errors that introduce security vulnerabilities, including buffer overflows, integer errors, and memory leaks.
   f. Discuss potential errors and security implications from both strong-type and weak-type languages.
   g. Evaluate the risks in using third-party applications, software tools, and libraries.
   h. Carry out a code review on a program component using a provided security checklist.
   i. Describe potential security vulnerabilities in event-driven GUI applications.

COURSE CONTENT: Topical areas of study include -

- Accessing data files and streams
- Inheritance and Polymorphism
- Arithmetic operators
- Integrated Development Environments
- Classes and objects
- Methods and functions
- Code Reviews
- Security
- Collections
- Software engineering
- Debugging
- Software maintenance
- Event-driven programming
- Software reuse
- Exception handling
- Standard Library
- Expressions and statements
- Strings
- Graphics and Graphical User Interfaces
- Variables/Constants

Course Addendum - Syllabus (Click to expand)