CMPS 161 – Algorithm Design and Implementation I

**Current Course Description:** Credit 3 hours. Prerequisite: Mathematics 161 or 165 or permission of the Department Head. Basic concepts of computer programming, problem solving, algorithm development, and program coding using a high-level, block structured language. Credit may be given for both Computer Science 110 and 161.

**Minimum Topics:**
- Fundamental programming constructs: Syntax and semantics of a higher-level language; variables, types, expressions, and assignment; simple I/O; conditional and iterative control structures; functions and parameter passing; structured decomposition
- Algorithms and problem-solving: Problem-solving strategies; the role of algorithms in the problem-solving process; implementation strategies for algorithms; debugging strategies; the concept and properties of algorithms
- Fundamental data structures: Primitive types; single-dimension and multiple-dimension arrays; strings and string processing
- Machine level representation of data: Bits, bytes and words; numeric data representation; representation of character data
- Human-computer interaction: Introduction to design issues
- Software development methodology: Fundamental design concepts and principles; structured design; testing and debugging strategies; test-case design; programming environments; testing and debugging tools

**Learning Objectives:** Students will be able to:
- Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.
- Modify and expand short programs that use standard conditional and iterative control structures and functions.
- Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.
- Choose appropriate conditional and iteration constructs for a given programming task.
- Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
- Describe the mechanics of parameter passing.
- Create algorithms for solving simple problems.
- Use pseudo code or a programming language to implement, test, and debug algorithms for solving simple problems.
- Describe some strategies that are useful in debugging.
- Discuss the representation and use of primitive data types and built-in data structures.
- Discuss the declaration, representation, and simple use (such as concatenation) of strings.
- Describe how scalars, strings, and arrays are allocated and used in memory.
- Write programs that use arrays (single-dimension and multi-dimension) and strings.
- Choose the appropriate data structure for modeling a given problem.
- Describe the ranges and memory requirements of different primitive data types
- Describe the internal representation of characters and strings for textual data
- Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
- Describe the phases of program translation from source code to executable code and the files produced by these phases.
- Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size.
- Discuss type incompatibility.
- Demonstrate the difference between call-by-value and call-by-reference parameter passing.
- Defend the importance of abstractions, especially with respect to programming-in-the-large.
- Discuss the properties of good software design.
- Select and apply appropriate design patterns in the construction of a software application.
- Demonstrate the capability to use a range of software tools in support of the development of a software product of medium size.

Relevance to Program Learning Outcomes and Evaluation:

a. An ability to apply knowledge of computing and mathematics appropriate to the discipline
   Justification: Students learn fundamental programming constructs such as sequence, selection, iteration and abstraction (use of methods), as well as fundamental data structures such as primitive types and arrays. Students learn conversion between binary, octal, decimal, and hexadecimal bases.
   Measured by: exams/quizzes and activities/programs throughout semester

b. An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution
   Justification: Students learn to analyze a program through successive refinement and top-down design, to divide large problems into smaller ones, and to determine the data objects and tools that will be needed to solve a problem.
   Measured by: exams/quizzes and activities/programs throughout semester

c. An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs
   Justification: Students design, implement, and test several programs in this course to meet the instructor's specifications.
   Measured by: exams/quizzes and activities/programs throughout semester

i. An ability to use current techniques, skills, and tools necessary for computing practice
   Justification: Students learn to use the JGrasp integrated development environment, along with the current version of the Java JDK. They are exposed to some rudimentary use of the symbolic debugger in JGrasp.
   Measured by: activities/programs throughout semester
Units covered:
PF1  Fundamental programming constructs (9/9)
PF2  Algorithms and problem-solving (3/6)
PF3  Fundamental data structures (2/14)
AR2  Machine level representation of data (1/3)
PL3  Introduction to language translation (1/2))
PL4  Declarations and types (1/3)
PL5  Abstraction mechanisms (1/3)
SE1  Software design 3 core hours (3/8)
SE3  Software tools and environments (2/3)