

***IV. Computer Science Courses and  
Recommended Courses of Study***

## **1. Foundations of Informational Technology (Computer Literacy)**

3–4 semester hours

Prerequisite: Intermediate Algebra and Geometry both with a “C” or better or appropriate placement

This course is designed to provide participants with a broad overview of computer concepts including key terminology and components of computer hardware, software, and operating systems and their use in problem solving.

Topics will include, but are not limited to, computer architecture, peripheral devices, networking components, system software, information system analysis, application software including word processing, database management, spreadsheet, and presentation software. Discussions will also include the Internet and web page development. This study of the uses and limitations of technology will lead to an informed decision about using computer resources.

This course is designed to fulfill general education requirements for a course in computing. It is not a prerequisite for any other college course and does not count as credit toward a degree in computer science. It does not meet the mathematics general education requirement.

### **Course Content**

1. History of computers and how computers can be categorized.
2. Computer components, including input, output, storage, and communication devices as well as how these devices help process data.
3. System unit components including the motherboard, central processing unit, memory, ports, expansion slots, and buses.
4. System software including various operating systems, utilities, and programming languages.
5. Information system classification and factors in choosing a system.
6. Electronic commerce.
7. Application and productivity software such as home and personal, educational and reference, graphics and communication, including applications that focus on word processing, spreadsheets, presentation, image processing, and databases.
8. Telecommunication and networks involving networking components, data transmission characteristics, and communication media.
9. Applications of the Internet, viewing web pages, navigating the Internet and searching for information, as well as web page development.
10. Programming concepts and languages.
11. Multimedia and artificial intelligence.

12. Security issues and strategies including network and Internet risks as well as hardware and software risks.
13. Computer ethics.
14. Information Technology careers, opportunities, preparation, certificates, and work environments.

### ***Learning Objectives***

1. Understand, and use appropriately, common computer technology.
2. Summarize significant historical events in the development of computer technologies as well as the future outlook of computers.
3. List the various hardware components of a common computer system and define the function and common characteristics of each.
4. Explain the capabilities and limitations of a computer as a medium for representing, storing, manipulating, and communicating various forms of information.
5. Demonstrate an understanding of the computational capability of information processing software including spreadsheets, data bases, word processors, presentation software, and graphics software.
6. Demonstrate an understanding of operating system software and skill in the use of at least one type of operating system software to perform system tasks such as creating and deleting files and performing routine maintenance functions.
7. Demonstrate skill in the use of the Internet for retrieving information.
8. Demonstrate an understanding of the development of Web pages.
9. State the means by which a computer transforms information from human terms to digital form through a high level programming language.
10. Demonstrate the ability to analyze real-world problems and select appropriate computer resources to solve them.
11. Define a general network and compare various types of computer networks.
12. Specify how computers impact virtually every aspect of society and present at least one way computers impact on each student's chosen career field.
13. Demonstrate an understanding of security issues and strategies.
14. Explain computer ethics and ethical guidelines as well as privacy and property protection.

## **2. Computer Science I**

3–4 semester hours

Prerequisite: Intermediate algebra and Geometry with a grade of “C” or better.

This course is designed to be the first course for those who wish to study computer science, including students who are majoring in computer science, mathematics, or engineering. Topics include the history and ethics of computer science, software life cycle, debugging, data types, variables, decision statements, loops, arrays, functions, input/output, data abstraction, and objects.

### **Course Content**

1. A brief introduction to the history of computer science starting with the first calculating machines to the beginning of electronic computing and modern technology.
2. Professional and public issues regarding use of computers including intellectual property.
3. Basic computer hardware including the central processing unit, read-only memory, random-access memory, input/output devices, and peripherals.
4. The different types of computer software: operating systems, high- and low-level languages, compilers, interpreters, clients, and servers.
5. The software life-cycle: problem definition, algorithm design, desktop testing, translation to a computer language, and debugging.
6. Appropriate indenting, comments, variable names, and function names.
7. Basic output to the console and input from the keyboard.
8. Types, variable declarations, and object instantiation.
9. Writing expressions using variables, constants, operators, object methods, and object properties.
10. Program flow control with conditionals, loops, and selection statements.
11. Using libraries including strings and mathematical functions.
12. \*Principles of graphical user interfaces; using GUI toolkits for input and output.
13. Program organization with functions and procedures including the use of parameters.
14. Introduction to recursion.
15. Introduction to object-oriented programming using data abstraction: inheritance, encapsulation, and polymorphism.

16. Single and two-dimensional arrays.
17. Input and output to text files.

### ***Learning Objectives***

1. Enumerate several milestones in the history of computing.
2. Describe the professional and ethical obligations of computer programmers.
3. Describe the components of a personal computer.
4. Differentiate between the different types of computer software.
5. Organize a programming problem by specifying the required task, developing routines, checking the routines with paper and pencil, translating it to code, and then debugging.
6. Write programs that are readable and easily maintained.
7. Create programs that send output to the console and read input from the keyboard.
8. Declare variables using predefined types.
9. Perform mathematical and logical operations on variables by writing expressions.
10. Use program control statements including if-then-else structures and loops.
11. Manipulate strings (for example, insert, concatenation, and substrings) using a standard string object.
12. Write expressions using mathematical functions from a standard library.
13. \*Building a simple graphical user interface for input and output.
14. Create functions and procedures and pass values to them using parameters.
15. Use local variables within functions and procedures.
16. Identify the recursive and base cases of a method using recursion and trace its execution.
17. Create user-defined objects.
18. Encapsulate an object's data and code so that other parts of the program will have only certain types of access to the object.
19. Use name overloading to create methods with different signatures.
20. Declare array variables and use them to display and manipulate lists of data.
21. Use a text stream to write to/read from disk.

\*This is optional material for courses using the C++ language.

### **3. Computer Science II**

3–4 semester hours

Prerequisite: Computer Science I with a grade of "C" or better.  
College Algebra with a grade of "C" or better.

This course is designed to be the second course for those who wish to study computer science, including students who are majoring in computer science, mathematics, or engineering. Topics include object-oriented programming, classes, an introduction to algorithms including basic searches and sorts, an introduction to dynamic data structures such as lists, stacks, queues and trees, event-driven programming with graphics, complexity analysis and recursion, records, and tables. Projects will be of a larger scale than in Computer Science I.

This course should use the same programming language as Computer Science I.

#### **Course Content**

1. Life-cycle of software: specification, design, risk analysis, verification, coding, testing, refining, production, and maintenance.
2. Modular program design using abstraction, object-oriented design, top-down design, design patterns, and modeling tools.
3. Appropriate program style that will simplify the task of the programmer and enhance the user's experience.
4. Implementation of small-sized to mid-sized projects.
5. Inheritance, class hierarchies, abstract classes, container classes, and iterators.
6. Polymorphism including examples of static binding and dynamic binding.
7. Fundamentals of event-driven programming including exception handling.
8. \*Graphics basics, graphics APIs, and graphical user interfaces (GUI).
9. Program verification by rigorous proof.
10. Elementary complexity analysis with big-O metrics.
11. Introduction to algorithms using basic searches and sorts, recursion, and text processing.
12. Survey of linear (lists, stacks, and queues) and non-linear (binary search trees and heaps) data structures.
13. List variations: header nodes, doubly linked lists, and circular lists.
14. Traversal, insertion, and deletion algorithms in lists, trees, and heaps.

15. Memory issues including dynamically and statically allocated memory, external versus internal, and memory leaks.
16. Records and tables are introduced as means of organizing data.
17. Storing structured data on disk for later retrieval.

### ***Learning Objectives***

1. Organize a programming problem by specifying the required task, developing routines, checking the routines with paper and pencil, translating it to code, and then debugging.
2. Write programs that are robust, easy to modify, read, maintain, and update.
3. Design abstract data types using inheritance, encapsulation, and polymorphism to solve complicated problems.
4. Use class hierarchies to identify the inherited methods and properties of classes.
5. Create a container class with its own API.
6. Use an iterator to access to contents of a container class.
7. Create code that will respond to user events including appropriate exception handling.
8. \*Use a standard graphical API to generate images to a canvas.
9. Use formal methods to verify the correctness of an algorithm including the use of mathematical induction.
10. Use big-O metrics to analyze elementary searching and sorting (selection, insertion, merge, quick, and heap) algorithms.
11. Program dynamic data handling routines using lists, stacks, queues, and trees.
12. Design an abstract data type that represents a linked list where methods are used to add, insert, delete, and display nodes.
13. Trace traversal, insertion, and deletion algorithms for linked list variations.
14. Use a binary search tree to quickly search and sort data.
15. Write recursive algorithms to traverse binary search trees and other simple tasks.
16. Use records to organize real world data sets.
17. Design tables to store information for later retrieval.

\*This is optional material for courses using the C++ language.

#### **4. Computer Science III**

3–4 semester hours

Prerequisite: Computer Science II with a grade of “C” or better.  
Discrete Mathematics with a grade of “C” or better.

This course is designed to be the third course for those who wish to study computer science, including students who are majoring in computer science, mathematics, or engineering. Topics include algorithms and algorithmic analysis, the design and implementation of data structures, hash tables, balanced binary search trees, graphs, recurrence relations, program complexity and efficiency, random number generation, and distributed methods. Projects will be of a larger scale than Computer Science II and will be team-based.

##### **Course Content**

1. Implementation of mid-sized projects using teams of programmers.
2. Survey of various sorting algorithms (insertion, shell, merge, heap, quick, and radix).
3. Review of basic data structures: lists, stacks, queues, trees, and heaps.
4. Tree variations: AVL trees and B-trees.
5. Advanced ADTs (sets, graphs, priority queues, and hash tables).
6. Hash functions and collision resolution.
7. Graph spanning tree algorithms, Kruskal's, and Dijkstra's algorithms.
8. Implementation strategies for determining which data structure is best for a given application.
9. Algorithmic paradigms (divide and conquer, greedy, dynamic, and backtracking).
10. Use of recursion to implement backtracking, depth-first, and breadth-first searching algorithms.
11. Designing algorithms with regards to the efficiency of time and memory.
12. Complexity analysis using big-O, big omega, big theta, and little-O.
13. Use of recursion in complexity.
14. Advanced sorting algorithms that combine internal and external techniques based upon memory needs.
15. Random number generation.
16. Message passing and distributed algorithms.

17. Domain decomposition and granularity.
18. Measuring the efficiency of distributed algorithms including speedup and overhead.

### ***Learning Objectives***

1. Build medium-sized projects in teams of programmers.
2. Use a binary search tree to quickly search and sort data.
3. Create a hash table and write an efficient hash function.
4. Implement various types of graphs, search them, and write them to disk.
5. Use Cayley's formula or Kirchhoff's Theorem to find the number of spanning trees.
6. Implement algorithms to find minimum spanning trees.
7. Find the minimal path between two vertices of a weighted graph.
8. Given a particular set of data, choose the best data type to represent the collection.
9. Create functions that can be called recursively to solve problems including constraint satisfaction problems and recursion on graphs.
10. Describe the complexity and efficiency of algorithms using big-O, big omega, big theta, and little-O metrics.
11. Solve recurrence relationships as a result of algorithmic analysis.
12. Implement an external sorting routine for large amounts of data.
13. Analyze a random number generator to determine its period of pseudo-random numbers.
14. Decompose an algorithm into independent processes that minimize communication needs.
15. Implement a message passing algorithm to concurrently communicate data using semaphores and barriers.
16. Analyze the execution time, speedups, and overhead of various distributed algorithms.

## **5. Computer Programming for Science and Engineering**

3–4 semester hours

Prerequisite: Calculus I with a grade of “C” or better

This course is designed to be a first course in programming of numerically intensive algorithms. Beyond a first course in computer programming, it should include a survey of introductory numerical methods. Topics include input, output and calculation of elementary data types, repetition and selection control structures, subprograms, and file and array processing. C++ or Java versions of the course should include class construction and object implementation. The addition of a computer algebra software tool can serve to complement the programming exercises with a mixture of symbolic and numeric enhancements.

Electrical and computer engineering generally require C, C++ or Java programming while other specialties prefer FORTRAN. The computer algebra material can be implemented in any of Mathematica, Maple, Derive or various Texas Instruments products (TI-89, TI-200, TI-Interactive).

### **Course Content**

1. Programming language
  - a. System software for editing, compiling, and executing programs.
  - b. Hardware overview and how it is related to data representation.
  - c. Debugging techniques in response to syntax and algorithmic errors.
  - d. Input and output commands for numeric (FORTRAN integer, real, double precision; C/C++/Java int, float, double) and character (FORTRAN character; C/C++/Java char) data types.
  - e. Arithmetic operations and assignment statements.
  - f. Choosing one of multiple options with selection statements.
  - g. Repeated execution of blocks of code.
  - h. Modularity with subprograms (FORTRAN functions, subroutines; C/C++ functions, methods; Java methods).
  - i. Intermediate data types (FORTRAN array, file; C array, file, struct; C++/Java array, file, class).
  - j. Introduction to object-oriented programming using data abstraction: inheritance, encapsulation, polymorphism. (C++/Java)
2. Numerical methods
  - a. Absolute and relative error analysis.
  - b. Root searching algorithms and their limitations (Binary search and Newton methods).
  - c. Derivative approximations at boundary and interior points (Forward, Central, and Backward approximations).
  - d. Quadrature methods to approximate definite integrals (Trapezoid and Simpson methods).
  - e. Integration methods to approximate differential equations (Euler and Runge-Kutte methods).
  - f. Linear regression applied to discrete data.

3. Computer algebra system
  - a. Symbolic and approximate calculations from algebra and calculus.
  - b. Visualization techniques applied to continuous and discrete data.
  - c. List operations including vector and matrix operations.
  - d. Regression libraries applied to discrete data.

### ***Learning Objectives***

1. Programming language & numerical methods
  - a. Compose, edit, and execute interactive programs performing input, output, and arithmetic operations on elementary numeric data types.
  - b. Design programs demonstrating control statements that perform selection and repetition on blocks of code.
  - c. Decompose algorithms into smaller segments using subprograms that communicate with parameters.
  - d. Use sequential access files for input and output of batch programs.
  - e. Demonstrate the manner in which arrays can be used to perform vector and matrix operations.
  - f. Use error analysis to ascertain accuracy and reliability of numerical results.
  - g. Apply iterative algorithms to search for roots of continuous functions.
  - h. Design accumulation algorithms that demonstrate methods of quadrature, integration, and/or linear regression.
  - i. Use files for input and output of regression points and/or matrix elements.
  - j. Create user-defined objects.
  - k. Encapsulate an object's data and code so that other parts of the program will have only certain types of access to the object.
2. Computer algebra system
  - a. Understand the use and limitations of exact and numeric calculations as applied to algebra, calculus, and matrix operations.
  - b. Visualize continuous functions and systems of equations.
  - c. Use list operations to manipulate discrete data.
  - d. Invoke regression library commands to model discrete cartesian points.
  - e. Visualize discrete data points along with regression models.

**Note:** If this course is to be used by an Engineering Computer Science Major as the first course of the normal sequence of Computer Science I-II, this course needs to be in the same language as Computer Science II.

## **6. *Discrete Structures***

3–4 semester hours

Prerequisite: College Algebra with a grade of “C” or better.

Refer to the description of Discrete Mathematics (page 47) of this guide.

## **7. Event Driven Programming**

3–4 semester hours

Prerequisite: Computer Science I with a grade of “C” or better

This course surveys event driven programming methods possessing Graphic User Interface (GUI) components that utilize Human Computer Interaction (HCI) screens for input. The intention is to keep this programming paradigm consistent, rather than in competition, with object-oriented paradigms. Additional topics include exception handling and an introduction to database manipulation.

The underlying assumption here is that the course is implemented in Java although other options such as Visual Basic and Javascript are available. The prerequisite allows the course to rapidly move through rudimentary programming practices to support the need to cover intermediate GUI construction and introductory database topics in the language.

### **Course Content**

1. Review of intermediate constructs from CS I (files, arrays, classes).
2. Modes of execution: batch versus interactive and applications appropriate for each mode.
3. Methods of User Interaction (UI): command-line versus event-driven.
4. Event models, event source, and event listener.
5. Event handlers and event propagation.
6. Exceptions and their handling; multiple exception processing; exception hierarchies.
7. HCI styles and techniques.
8. Aspects of screen design: layout, color, fonts, and labeling.
9. Responsibilities of UI versus the application.
10. Language tools available for building a GUI; geometry management; mouse event monitors.
11. Widgets to aid in the processing of input/output: buttons, boxes, labels, text, and windows.
12. Network security management: authentication protocols, digital signatures, and digital certificates.
13. Introduction to Structured Query Language (SQL): select, insert, delete, and update queries.
14. Database objects and connectivity techniques.

## ***Learning Objectives***

1. Continue the development of object-oriented programs to produce maintainable applications.
2. Explain the difference between event-driven programming and command-line programming.
3. Understand the role that event handlers play between event sources and event targets.
4. Develop code that responds to exception conditions raised during execution.
5. Choose screen aspects to aid in HCI and techniques.
6. Utilize language layout managers to arrange screen fields.
7. Explain good design principles of widgets; sequenced screen presentations; and simple error-trap dialog.
8. Design, code, test, and debug simple event-driven programs that respond to user events such as completed input fields or mouse activity.
9. Describe methods used to create a secure connection to a data source.
10. Use basic SQL commands within an application.
11. Create applications that display, insert, modify, and delete data from a database.

## **8. Computer Organization and Architecture**

3–4 semester hours

Prerequisite: Computer Science I with a grade of “C” or better

This course is a survey of the various levels of hierarchical computer architecture and design. The analysis of internal and external memory models, busses, I/O peripherals, complex instruction set computer (CISC), and reduced instruction set computer (RISC) processor strategies are covered. Additional topics include the instruction formats and addressing schemes of microprocessors such as Intel Pentium and Power PC architectures, vectorizing multiprocessors, and multi-computer systems.

Rather than a course in assembler language programming, rudimentary programming assignments in the language can be used to demonstrate aspects of computer architecture. Multi-computer and multiprocessor programming assignments can be demonstrated using environments such as MPI.

### **Course Content**

1. A historical overview on how computer architectures have evolved including economic trends that have motivated the need for underlying technologies.
2. Computer performance measured in popular metrics: MIPS, MFLOPS, GFLOPS, and TFLOPS.
3. Integer data representation in various bases (binary, octal, and hexadecimal) and the means of performing arithmetic operations in the binary system.
4. Floating point numeric representation including normalization and IEEE standards.
5. The use of gates to represent elementary arithmetic and logical operations.
6. The program instruction cycle with fetch, execute, and interrupt activities.
7. Interconnection of components with busses demonstrating typical configurations (ISA, PCI) and bandwidths.
8. Hierarchical memory systems and their impact on performance, the principle of locality, caches, main memory, and virtual and external memory (disk, tape) strategies.
9. Input/Output (I/O) performance measures, types and characteristics of devices including the connections of I/O devices to processor, memory, and operating systems.

10. Arithmetic logic units, their construction, operations, and operands.
11. Instruction set design, operations in the instruction set, type and size of operands, and instruction representation.
12. Pipelining instruction data paths, pipelined control, addressing hazards of branching and dependencies, and exploiting more instruction-level parallelism.
13. Taxonomies of symmetric multiprocessors and distributed memory multi-computers, tightly coupled non-uniformed memory access systems, loosely coupled clusters, and memory access and cache coherence strategies.

### ***Learning Objectives***

1. Review the historical development of computers and computer architectures.
2. Apply the basic metrics by which new and existing computer systems may be evaluated to systems used in the course.
3. Understand information representation, error detection/correction schemes, and digital logic.
4. Describe elementary computer arithmetic operations for integer and floating point data in binary.
5. Identify the basic components of computer organization and understand how they work together.
6. Survey the hierarchical internal and external memory organization strategies and understand their impact upon performance.
7. Become knowledgeable about the design of I/O modules, control units, and arithmetic logic units.
8. Learn the format of instruction sets, addressing modes, and the operation of the instruction cycle.
9. Understand the relative strengths and weaknesses of CISC and RISC architectures.
10. Demonstrate the use of pipelining and vector processing including data path and dependencies.
11. Recognize current superscalar microprocessor and multiprocessor models in today's market.

**9. Computer Science Major**

Recommended Course of Study: Associate of Science  
Major in Computer Science

For the student <u>ready</u> for Calculus	
<b>1. Freshman 1</b>	<b>3. Sophomore 1</b>
A. Computer Science I B. Calculus I C. Discrete Mathematics	A. Computer Science III B. Computer Organization and Architecture
<b>2. Freshman 2</b>	<b>4. Sophomore 2</b>
A. Computer Science II B. Calculus II	A. Event Driven Programming

For the student <u>not ready</u> for calculus	
<b>1. Freshman 1</b>	<b>3. Sophomore 1</b>
A. Computer Science I B. Precalculus	A. Computer Science III B. Computer Organization and Architecture C. Calculus I
<b>2. Freshman 2</b>	<b>4. Sophomore 2</b>
A. Computer Science II B. Discrete Mathematics	A. Event Driven Programming B. Calculus II

**Note:** The specific curriculum of study should be planned with appropriate advisors in accordance with specific requirements of the university to which students wish to transfer.

A student entering this program of study not adequately prepared to begin with the suggested courses should plan to enroll in the correct prerequisite courses and should plan to enroll in additional semesters in order to complete the expectations of this curriculum.

## **10. Engineering Computer Science Major**

Recommended course of study: Associate of Science

<b>1. Freshman 1</b>	<b>3. Sophomore 1</b>
A. Calculus I B. Discrete Mathematics	A. Computer Science II B. Calculus III C. Computer Organization & Architecture
<b>2. Freshman 2</b>	<b>4. Sophomore 2</b>
A. Computer Programming for Science and Engineering B. Calculus II	A. Computer Science III B. Linear Algebra and/or D. Differential Equations

**Note:** The specific curriculum of study should be planned with appropriate advisors in accordance with specific requirements of the university to which students wish to transfer.

A student entering this program of study not adequately prepared to begin with the suggested courses should plan to enroll in the correct prerequisite courses and should plan to enroll in additional semesters in order to complete the expectations of this curriculum.

**11. Mathematics, Physical Science, or Engineering (Mechanical, Industrial, Electrical) Major**

Recommended course of study: Associate of Science

<b>1. Freshman 1</b>	<b>3. Sophomore 1</b>
A. Calculus I	A. Computer Organization and Architecture B. Calculus III
<b>2. Freshman 2</b>	<b>4. Sophomore 2</b>
A. Computer Programming for Science and Engineering B. Calculus II	A. Linear Algebra and/or B. Differential Equations

**Note:** The specific curriculum of study should be planned with appropriate advisors in accordance with specific requirements of the university to which students wish to transfer.

A student entering this program of study not adequately prepared to begin with the suggested courses should plan to enroll in the correct prerequisite courses and should plan to enroll in additional semesters in order to complete the expectations of this curriculum.

## **12. Business Curriculums**

Recommended course of study: Associate of Arts

<b>1. Freshman 1</b>	<b>3. Sophomore 1</b>
A. Foundations of Information Technology B. College Algebra	A. Computer Science II B. Finite Mathematics
<b>2. Freshman 2</b>	<b>4. Sophomore 2</b>
A. Computer Science I B. Business Calculus	A. Event Driven Programming B. Business Statistics

**Note:** The specific curriculum of study should be planned with appropriate advisors in accordance with specific requirements of the university to which students wish to transfer.

A student entering this program of study not adequately prepared to begin with the courses suggested as the entry points should take the appropriate prerequisite courses and should plan to enroll in additional semesters in order to complete the expectations of this curriculum.

***V. Additional Course Options  
as Recommended by the Association of  
Computing Machinery (ACM)***

Illinois Articulation Initiative (IAI) Panel on Computer Science has defined the content of Computer Science courses that will be acceptable for transfer credit for the State of Illinois; whereas The Association for Computing Machinery (ACM) has recommended course content on a national level. The IAI Panel referred to the recommendations of the Two-Year College Education Committee of ACM in developing their transfer guidelines. A complete exposition of IAI can be found at URL [www.itransfer.org](http://www.itransfer.org) and a complete exposition of the ACM recommendations for Two-Year Computer Science programs can be found at the URL <http://acmtyc.org>.

Each organization has recommended the content for the Computer Science I – Computer Science III courses, which is very similar other than the order and the manner topics are covered. As an attempt to inform IMACC members of the ACM recommendations and to generate further discussion on the issues of content, prerequisites, order of content, manner of delivery, and objectives for the Computer Science I – Computer Science III courses, the following course descriptions will offer a different route to accomplish the course content of the courses that are currently approved by the IAI Computer Science Major Panel. The Net-Centric Operating Systems course description has been included to offer a new possibility for elective option.

It is important to note that the courses that follow are not approved by the IAI Computer Science Major Panel as equivalent or replacements for any of the IAI computer science courses; however as the computer science courses change with time, these course descriptions may prove helpful as a guide.

## ***Computer Science I–Version A***

3–4 semester hours

Prerequisite: Intermediate Algebra and Geometry with a grade of “C” or better in each.

This course is designed to be the first course for those who wish to study computer science, including students who are majoring in computer science, mathematics, or engineering. Topics include the history and ethics of computer science, software life cycle, debugging, data types, variables, decision statements, loops, arrays, functions, input/output, data abstraction, and objects.

### ***Course Content***

1. A brief introduction to the history of computer science starting with the first calculating machines to the beginning of electronic computing and modern technology.
2. Professional and public issues regarding use of computers including intellectual property.
3. Basic computer hardware including the central processing unit, read-only memory, random-access memory, input/output devices, and peripherals.
4. The different types of computer software: operating systems, high- and low-level languages, compilers, interpreters, clients, and servers.
5. The software life-cycle: problem definition, algorithm design, desktop testing, translation to a computer language, debugging.
6. Appropriate indenting, comments, variable names, and function names.
7. Basic output to the console and input from the keyboard.
8. Types, variable declarations, and object instantiation.
9. Writing expressions using variables, constants, operators, object methods, and object properties.
10. Program flow control with conditionals, loops, and selection statements.
11. Program organization with functions and procedures including the use of parameters.
12. Single and two-dimensional arrays.
13. Using libraries including strings and mathematical functions.
14. Input and output to text files.
15. Introduction to object-oriented programming using data abstraction: inheritance, encapsulation, polymorphism.

## ***Learning Objectives***

1. Enumerate several milestones in the history of computing.
2. Discuss current trends in computer science.
3. Describe the professional obligations of computer programmers and the role of codes of conduct.
4. Identify the rights of copyright and patent holders and the consequences of infringement upon those rights.
5. List various types of computer crime and explain how the software can be designed to combat it.
6. Describe the components of a personal computer.
7. Differentiate between the different types of computer software.
8. Organize a programming problem by specifying the required task, developing routines, checking the routines with paper and pencil, translating it to code, and then debugging.
9. Write programs that are readable and easily maintained.
10. Create programs that send output to the console and read input from the user.
11. Instantiate common objects (such as strings), call their methods, and access their properties.
12. Declare variables using predefined types.
13. Perform mathematical and logical operations on variables by writing expressions.
14. Use program control statements including if-then-else structures and loops.
15. Create functions and procedures and pass values to them using parameters.
16. Use local variables within functions and procedures.
17. Declare array variables and use them to display and manipulate lists of data.
18. Use a text stream to write to/read from disk.
19. Manipulate strings (for example, insert, concatenation, substrings) using a standard string object.
20. Write expressions using mathematical functions from a standard library.
21. Create user-defined objects.
22. Define objects that inherit properties and methods from other objects.

23. Encapsulate an object's data and code so that other parts of the program will have only certain types of access to the object.
24. Use polymorphism to create objects that have a single interface for different but similar tasks.

## ***Computer Science II–Version A***

3–4 semester hours

Prerequisite: Computer Science I with a grade of “C” or better.

This course is designed to be the second course for those who wish to study computer science, including students who are majoring in computer science, mathematics, or engineering. Topics include object-oriented programming, classes, introduction to algorithms including basic searches and sorts, introduction to dynamic data structures such as stacks and queues, event-driven programming, graphics, virtual machines, records, tables, and memory issues. Projects will be of a larger scale than in Computer Science I.

This course should use the same programming language as Computer Science I.

### ***Course Content***

1. Life-cycle of software: specification, design, risk analysis, verification, coding, testing, refining, production, and maintenance.
2. Modular program design is achieved using abstraction, object-oriented design, top-down design, design patterns, and modeling tools.
3. Appropriate program style is emphasized that will simplify the task of the programmer and enhance the user’s experience.
4. Implementation of small- to mid-sized projects.
5. Program verification.
6. Abstract data types: more advanced examples of inheritance, encapsulation, and polymorphism than found in Computer Science I.
7. Advanced class topics including class hierarchies; abstract, interface, and container classes; and iterators.
8. Introduction to algorithms using basic searches and sorts, recursion, and text processing.
9. Introduction to dynamic data structures such as sets, stacks, queues, and graphs.
10. Fundamentals of event-driven programming including exception handling.
11. \*Graphics basics, graphics APIs, graphical user interfaces (GUI), color models, and the mathematics of graphics.
12. Virtual machines including the hierarchy of virtual machines and intermediate languages.
13. Memory issues including dynamically and statically allocated memory, external versus internal, and memory leaks.

14. Records and tables are introduced as means of organizing data.
15. Storing structured data on disk for later retrieval.

### ***Learning Objectives***

1. Organize a programming problem by specifying the required task, developing routines, checking the routines with paper and pencil, translating it to code, and then debugging.
2. Select an appropriate design and implement it in a programming solution.
3. Write programs that are easy to modify, read, maintain, and update.
4. Write programs that are easy to use, well-documented, and catch user errors.
5. Use formal methods to verify the correctness of an algorithm including the use of mathematical induction.
6. Design abstract data types using inheritance, encapsulation, and polymorphism to solve complicated problems.
7. Use class hierarchies to identify the inherited methods and properties of classes.
8. Create a container class with its own API.
9. Use an iterator to access to contents of a container class.
10. Use elementary searches (like the binary search) and sorts (such as the selection sort).
11. Write recursive algorithms to solve simple tasks.
12. Program dynamic data handling routines using stacks and queues.
13. Create code that will respond to user events including appropriate exception handling.
14. \*Use a standard graphical API to generate images to a canvas.
15. Perform mathematical transformations to produce animation.
16. Explain what a virtual machine is and how it is used to make a program portable.
17. Compare the positive and negative aspects of compilation and interpretation.
18. Manage memory usage so that distributed solutions are able to execute on various machines.
19. When appropriate, write garbage collection routines that prevent the program from crashing the operating system.

20. Use records to organize real world data sets.
21. Design tables to store information for later retrieval.

\*This is optional material for courses using the C++ language.

## ***Computer Science III–Version A***

3–4 semester hours

Prerequisite: Computer Science II with a grade of “C” or better.  
Recommended: Discrete Mathematics with a grade of “C” or better.

This course is designed to be the third course for those who wish to study computer science, including students who are majoring in computer science, mathematics, or engineering. Topics include algorithms and algorithmic analysis, the design and implementation of data structures, heaps, hash tables, linked lists, graphs and trees, recursion, advanced searches and sorts, program complexity and efficiency, and random number generation. Projects will be of a larger scale than Computer Science II and will be team-based.

### ***Course Content***

1. Implementation of mid-sized projects using teams of programmers.
2. Designing algorithms with efficiency in mind.
3. Heap implementation including the delete-max, delete-min, decrease-key, insert, and merge operations.
4. Looking up information in hash tables.
5. Linked Lists: adding, inserting, deleting, and displaying nodes.
6. Trees and other types of graphs such as binary search trees, spanning trees, linear orders, and partial orders.
7. Implementation strategies for determining which data structure is best for a given application.
8. Algorithms involving graphs such as transversals, shortest-path algorithms, transitive closure, and minimal spanning trees.
9. In-depth study of recursion including backtracking and recursion on graphs.
10. Advanced sort and search algorithms.
11. Analyzing program complexity and determining best and worst case scenarios.
12. Measuring the efficiency of algorithms including execution times, algorithm growth rates, big-O and other notation.
13. Random number generation.

## ***Learning Objectives***

1. Build medium-sized projects in teams of programmers.
2. Work with heaps using the heap sort or selection algorithms.
3. Create a hash table and write an efficient hash function.
4. Design an abstract data type that represents a linked list where methods are used to add, insert, delete, and display nodes.
5. Implement various types of graphs, search them, and write them to disk.
6. Use a binary search tree to quickly search and sort data.
7. Use Cayley's formula or Kirchhoff's Theorem to find the number of spanning trees.
8. Analyze algorithms for find minimum spanning trees.
9. Given a particular set of data, choose the best data type to represent the collection.
10. For any partial order find an algorithm that will select every element exactly once.
11. Use algorithms to find a path between two vertices of a graph that has minimum weight.
12. Given an arbitrary relation, find the smallest transitive relation that contains it.
13. Create functions that can be called recursively to solve problems including constraint satisfaction problems and recursion on graphs.
14. Write search and sort routines that are the most efficient for a given problem.
15. Analyze the execution time of various algorithms.
16. Describe the complexity and efficiency of an algorithm using big-O, big omega, big theta, and little-O notation.
17. Analyze a random number generator to determine its period of pseudo-random numbers.

## ***Net Centric Operating Systems***

3–4 semester hours

Prerequisite: Computer Science II with a grade of “C” or better

This course introduces the fundamentals of operating systems design and implementation. Topics include an overview of the components of an operating system, mutual exclusion and synchronization, implementation of processes, scheduling algorithms, memory management, and file systems. It introduces the structured, implementation, and the theoretical underpinnings of computer networking and the applications enabled by that technology.

This course could contain an integrated laboratory component using a recent version of Linux, Unix, or Microsoft Windows operating systems and the Java programming language. Emphasis should be placed on the built-in networking and security capabilities of modern operating systems. Simple and secure network socket scripting can be accomplished with built-in methods from the Java networking class.

### ***Course Content***

1. Concurrent execution; states and state diagrams, implementation structures (ready lists, process control blocks, etc.); dispatching and context switching; and interrupt handling.
2. Mutual exclusion; deadlock detection, prevention; solution strategies; models and mechanisms (semaphores, monitors, condition variables, and rendezvous); producer-consumer problems; synchronization; and multiprocessor issues.
3. Preemptive and non-preemptive scheduling; scheduling policies; processes and threads; and real-time issues.
4. Physical memory and memory management hardware; overlays, swapping, and partitions; paging and segmentation; page placement and replacement policies; working sets and thrashing; and caching.
5. Fundamental file system concepts (data, metadata, operations, organization, buffering, and sequential vs. nonsequential files); content and structure of directories; file system techniques (partitioning, mounting and unmounting, and virtual file systems); memory-mapped files; special-purpose file systems; naming, searching, and access; and backup strategies.
6. Overview of system security; policy/mechanism separation; security methods and devices; protection, access, and authentication; models of protection; memory protection; encryption; and recovery management.
7. Network standards and standardization bodies; the ISO 7-layer reference model in general and its instantiation in TCP/IP; circuit switching and packet switching; streams and datagrams; physical layer networking concepts; data link layer concepts; Internetworking and routing; and transport layer services.

8. Protocols at the web application layer; principles of web engineering web sites; remote procedure calls; lightweight distributed objects; the role of middleware; support tools; security issues in distributed object systems; and enterprise-wide web-based applications.
9. Issues of network management; issues for Internet Service Providers (ISPs); security issues and firewalls; and quality of service issues.
10. Basic data compression; audio compression and decompression; image compression and decompression; video compression and decompression; and performance issues.
11. Multimedia data technologies; multimedia standards; capacity planning and performance issues; input and output devices; MIDI keyboards, synthesizers; storage standards; multimedia servers and file systems; and tools to support multimedia development.
12. Introduction to LANs and WANs; layered protocol design, ISO/OSI, IEEE 802; impact of architectural issues on distributed algorithms; network computing; and distributed multimedia.

### ***Learning Objectives***

1. Understand concurrent execution and the synchronization mechanisms available to avoid deadlock or congestion control.
2. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes.
3. Differentiate the mechanisms used in interfacing a range of devices (including hand-held devices, networks, and multimedia) to a computer.
4. Compare and contrast different approaches to file organization, including distributed file systems.
5. Recognize the current network standards and their impact upon switching and layer strategies.
6. Understand the basic concepts of network security including authentication, integrity, key distribution, and system security design challenges.
7. Utilize Web applications to demonstrate an example of client-server computing.
8. Design scripts with middleware tools used in distributed operating systems such as DCOM or CORBA.
9. Summarize the principles of virtual memory, caching, paging, and segmentation.
10. Compare various compression and decompression algorithms for different data types, such as text, graphics, sound, and video.

11. Understand the impact that data compression has on network traffic.
12. Recognize the current architectures for networks and distributed systems