

Fundamental Computer Science Concepts

A. Overview

Introduction to the discipline of computer science; covers the material traditionally found in courses that introduce problem solving, computer programming, algorithms, data structures, computer architecture, assembly programming and discrete structures.

B. Prerequisites and/or co-requisites: Pre-calculus with a grade of “C” or better.

C. Minimum units: 12 semester units

D. Course Topics and Learning Outcomes

Discrete Structures (DS)

DS1. Functions, relations, and sets

Minimum coverage time: 6 hours

Topics:

- Functions (surjections, injections, inverses, composition)
- Relations (reflexivity, symmetry, transitivity, equivalence relations)
- Sets (Venn diagrams, complements, Cartesian products, power sets)
- Pigeonhole principle
- Cardinality and countability

Learning outcomes:

1. Explain with examples the basic terminology of functions, relations, and sets.
2. Perform the operations associated with sets, functions, and relations.
3. Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
4. Demonstrate basic counting principles, including uses of diagonalization and the pigeonhole principle.

DS2. Basic logic

Minimum coverage time: 10 hours

Topics:

- Propositional logic
- Logical connectives
- Truth tables
- Normal forms (conjunctive and disjunctive)
- Validity
- Predicate logic
- Universal and existential quantification
- Modus ponens and modus tollens
- Limitations of predicate logic

Learning outcomes:

1. Apply formal methods of symbolic propositional and predicate logic.
2. Describe how formal tools of symbolic logic are used to model algorithms and real-life situations.
3. Use formal logic proofs and logical reasoning to solve problems such as puzzles.
4. Describe the importance and limitations of predicate logic.

DS3. Proof techniques

Minimum coverage time: 12 hours

Topics:

- Notions of implication, converse, inverse, contrapositive, negation, and contradiction
- The structure of formal proofs
- Direct proofs
- Proof by counterexample
- Proof by contraposition
- Proof by contradiction
- Mathematical induction
- Strong induction
- Recursive mathematical definitions
- Well orderings

Learning outcomes:

1. Outline the basic structure of and give examples of each proof technique described in this unit.
2. Discuss which type of proof is best for a given problem.
3. Relate the ideas of mathematical induction to recursion and recursively defined structures.
4. Identify the difference between mathematical and strong induction and give examples of the appropriate use of each.

DS4. Basics of counting

Minimum coverage time: 5 hours

Topics:

- Counting arguments
- Sum and product rule
- Inclusion-exclusion principle
- Arithmetic and geometric progressions
- Fibonacci numbers
- The pigeonhole principle
- Permutations and combinations
- Basic definitions
- Pascal's identity
- The binomial theorem
- Solving recurrence relations
- Common examples

Learning outcomes:

1. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
2. Solve a variety of basic recurrence equations.
3. Analyze a problem to create relevant recurrence equations or to identify important counting questions.

DS5. Trees

Minimum coverage time: 3 hours

Topics:

- Trees and Graphs
- Spanning trees
- Traversal strategies

Learning outcomes:

1. Demonstrate different traversal methods for trees and graphs.
2. Model problems in computer science using trees.
3. Relate trees to data structures, algorithms, and counting.

Programming Fundamentals (PF)

PF1. Fundamental programming constructs

Minimum coverage time: 9 hours

Topics:

- Basic 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

Learning outcomes:

1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.
2. Modify and expand short programs that use standard conditional and iterative control structures and functions.
3. 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.
4. Choose appropriate conditional and iteration constructs for a given programming task.
5. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
6. Describe the mechanics of parameter passing.

PF2. Algorithms and problem-solving

Minimum coverage time: 6 hours

Topics:

- Problem-solving strategies
- The role of algorithms in the problem-solving process
- Implementation strategies for algorithms
- Debugging strategies
- The concept and properties of algorithms

Learning outcomes:

1. Discuss the importance of algorithms in the problem-solving process.
2. Identify the necessary properties of good algorithms.
3. Create algorithms for solving simple problems.
4. Use pseudocode or a programming language to implement, test, and debug algorithms for solving simple problems.
5. Describe strategies that are useful in debugging.

PF3. Fundamental data structures

Minimum coverage time: 12 hours

Topics:

- Primitive types
- Arrays
- Records
- Strings and string processing
- Data representation in memory
- Static, stack, and heap allocation
- Runtime storage management
- Pointers and references
- Linked structures
- Implementation strategies for stacks, queues, and hash tables
- Implementation strategies for trees
- Strategies for choosing the right data structure

Learning outcomes:

1. Discuss the representation and use of primitive data types and built-in data structures.
2. Describe how the data structures in the topic list are allocated and used in memory.
3. Describe common applications for each data structure in the topic list.
4. Implement the user-defined data structures in a high-level language.
5. Compare alternative implementations of data structures with respect to performance.
6. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables.
7. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
8. Choose the appropriate data structure for modeling a given problem.

PF4. Recursion

Minimum coverage time: 5 hours

Topics:

- The concept of recursion
- Recursive mathematical functions
- Simple recursive procedures
- Divide-and-conquer strategies
- Recursive backtracking
- Implementation of recursion

Learning outcomes:

1. Describe the concept of recursion and give examples of its use.
2. Identify the base case and the general case of a recursively defined problem.
3. Compare iterative and recursive solutions for elementary problems such as factorial.
4. Describe the divide-and-conquer approach.
5. Implement, test, and debug simple recursive functions and procedures.
6. Describe how recursion can be implemented using a stack.
7. Discuss problems for which backtracking is an appropriate solution.
8. Determine when a recursive solution is appropriate for a problem.

Architecture and Organization (AR)

AR2. Machine level representation of data

Minimum coverage time: 3 hours

Topics:

- Bits, bytes, and words
- Numeric data representation and number bases
- Fixed- and floating-point systems
- Signed and twos-complement representations
- Representation of nonnumeric data (character codes, graphical data)
- Representation of records and arrays

Learning outcomes:

1. Explain the reasons for using different formats to represent numerical data.
2. Explain how negative integers are stored in sign-magnitude and twos complement representation.
3. Convert numerical data from one format to another.
4. Discuss how fixed-length number representations affect accuracy and precision.
5. Describe the internal representation of nonnumeric data.
6. Describe the internal representation of characters, strings, records, and arrays.

AR3. Assembly level machine organization

Minimum coverage time: 9 hours

Topics:

- Basic organization of the von Neumann machine
- Control unit; instruction fetch, decode, and execution
- Instruction sets and types (data manipulation, control, I/O)
- Assembly/machine language programming
- Instruction formats
- Addressing modes
- Subroutine call and return mechanisms
- I/O and interrupts

Learning outcomes:

1. Explain the organization of the classical von Neumann machine and its major functional units.
2. Explain how an instruction is executed in a classical von Neumann machine.
3. Summarize how instructions are represented at both the machine level and in the context of a symbolic assembler.
4. Explain different instruction formats, such as addresses per instruction and variable length vs. fixed length formats.
5. Write simple assembly language program segments.
6. Demonstrate how fundamental high-level programming constructs are implemented at the machine-language level.
7. Explain how subroutine calls are handled at the assembly level.
8. Explain the basic concepts of interrupts and I/O operations.

Programming Languages (PL)

PL1. Overview of programming languages

Minimum coverage time: 2 hours

Topics:

- History of programming languages
- Brief survey of programming paradigms
- Procedural languages
- Object-oriented languages
- Functional languages
- Declarative, non-algorithmic languages

- Scripting languages
- The effects of scale on programming methodology

Learning outcomes:

1. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today.
2. Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit.
3. Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
4. Distinguish between programming-in-the-small and programming-in-the-large.

PL4. Declarations and types

Minimum coverage time: 3 hours

Topics:

- The conception of types as a set of values together with a set of operations
- Declaration models (binding, visibility, scope, and lifetime)
- Overview of type-checking
- Garbage collection

Learning outcomes:

1. Explain the value of declaration models, especially with respect to programming-in-the-large.
2. Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size.
3. Discuss type incompatibility.
4. Demonstrate different forms of binding, visibility, scoping, and lifetime management.
5. Defend the importance of types and type-checking in providing abstraction and safety.
6. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection).

PL5. Abstraction mechanisms

Minimum coverage time: 3 hours

Topics:

- Procedures, functions, and iterators as abstraction mechanisms
- Parameterization mechanisms (reference vs. value)
- Activation records and storage management
- Type parameters and parameterized types - templates or generics
- Modules in programming languages

Learning outcomes:

1. Explain how abstraction mechanisms support the creation of reusable software components.
2. Demonstrate the difference between call-by-value and call-by-reference parameter passing.
3. Defend the importance of abstractions, especially with respect to programming-in-the-large.

4. Describe how the computer system uses activation records to manage program modules and their data.

PL6. Object-oriented programming

Minimum coverage time: 10 hours

Topics:

- Object-oriented design
- Encapsulation and information-hiding
- Separation of behavior and implementation
- Classes and subclasses
- Inheritance (overriding, dynamic dispatch)
- Polymorphism (subtype polymorphism vs. inheritance)
- Class hierarchies
- Collection classes and iteration protocols
- Internal representations of objects and method tables

Learning outcomes:

1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism.
2. Design, implement, test, and debug simple programs in an object-oriented programming language.
3. Describe how the class mechanism supports encapsulation and information hiding.
4. Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance.
5. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
6. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
7. Describe how iterators access the elements of a container.

Social and Professional Issues (SP)

SP1. History of computing

Minimum coverage time: 1 hour

Topics:

- Prehistory -- the world before 1946
- History of computer hardware, software, networking
- Pioneers of computing

Learning outcomes:

1. List the contributions of several pioneers in the computing field.
2. Compare daily life before and after the advent of personal computers and the Internet.
3. Identify significant continuing trends in the history of the computing field.

Software Engineering (SE)

SE1. Software design

Minimum coverage time: 8 hours

Topics:

- Fundamental design concepts and principles
- Design strategy
- Software architecture
- Structured design

- Object-oriented analysis and design
- Component-level design
- Design for reuse

Learning outcomes:

1. Discuss the properties of good software design.
2. Compare and contrast object-oriented analysis and design with structured analysis and design.
3. Evaluate the quality of multiple software designs based on key design principles and concepts.
4. Select and apply appropriate design patterns in the construction of a software application.
5. Create and specify the software design for a software product from a software requirement document, an accepted program design methodology (e.g., structured or object-oriented), and appropriate design notation.
6. Conduct a software design review using appropriate guidelines.
7. Evaluate a software design at the component level.
8. Evaluate a software design from the perspective of reuse.

SE2. Using APIs

Minimum coverage time: 5 hours

Topics:

- API programming
- Class browsers and related tools
- Programming by example
- Debugging in the API environment
- Introduction to component-based computing

Learning outcomes:

1. Explain the value of application programming interfaces (APIs) in software development.
2. Use class browsers and related tools during the development of applications using APIs.
3. Design, implement, test, and debug programs that use large-scale API packages.

AL: Algorithms

Minimum coverage time: 2 hours

Topics:

- Big O notation
- Basic sorting and searching algorithms

Learning outcomes:

1. Explain the use of O notation.
2. Explain the basic sorting and searching algorithms.

E. Justification

1. This 12 semester unit description has been approved by the CS group both at the IMPAC and the LDTP meetings.
2. The 12 semester unit described introduces students to a set of fundamental computer science topics. In the spirit of the ACM 2001 Curriculum Report, from which these topic groups are based, the disciplinary group is not specifying individual course content.

3. Most Community Colleges have an Introduction to Programming, Programming with Data Structures, Discrete Mathematics, Assembly/Architecture courses which would articulate after some modifications to their courses.
4. Individual institutions have the freedom, as they believe appropriate to package the material in any number of courses.