

Fundamentals of Object-Oriented Programming

(adapted from the Final Report of the Computing Curricula 2001 project,
a joint undertaking of the IEEE-CS and the ACM)

Fluency in a programming language is prerequisite to the study of most of computer science. Students entering a graduate computer science programs must know how to use at least one programming language well; and ideally students are competent in languages that make use of at least two programming paradigms. This knowledge area consists of those skills and concepts that are essential to programming practice according to the object-oriented paradigm, though the early material is actually independent of a paradigm. As a result, this area includes units on fundamental programming concepts, basic data structures, and algorithmic processes. These units, however, by no means cover the full range of programming knowledge that an entering computer science graduate student must know.

Topics:

1. Basic syntax and semantics of a higher-level language
2. Variables, types, expressions, and assignment
3. Simple I/O
4. Conditional and iterative control structures
5. Functions and parameter passing
6. Structured decomposition
7. Debugging strategies
8. The concept and properties of algorithms
9. Primitive types (integers, reals, characters, boolean)
10. Arrays
11. Strings and string processing
12. Pointers and references
13. Linked structures
14. Implementation strategies for stacks and queues
15. The concept of recursion
16. Recursive mathematical functions
17. Simple recursive procedures
18. Divide-and-conquer strategies
19. Exception handling
20. Object-oriented design
21. Encapsulation and information-hiding
22. Separation of behavior and implementation
23. Classes and subclasses
24. Inheritance (overriding, dynamic dispatch)
25. Polymorphism (subtype polymorphism vs. inheritance)
26. Class hierarchies
27. Collection classes and iteration protocols
28. Sequential and binary search algorithms
29. Quadratic sorting algorithms (selection, insertion)

Learning objectives:

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.
7. Create algorithms for solving simple problems.
8. Describe strategies that are useful in debugging.
9. Write programs that use each of the following data structures: arrays, strings, linked lists, stacks, and queues
10. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
11. Describe the concept of recursion and give examples of its use.
12. Identify the base case and the general case of a recursively defined problem.
13. Compare iterative and recursive solutions for elementary problems such as factorial.
14. Describe the divide-and-conquer approach.
15. Implement, test, and debug simple recursive functions and procedures.
16. Explain the difference between event-driven programming and command-line programming.
17. Design, code, test, and debug simple event-driven programs that respond to user events.
18. Develop code that responds to exception conditions raised during execution.
19. Design, implement, test, and debug simple programs in an object-oriented programming language.
20. Describe how the class mechanism supports encapsulation and information hiding.
21. Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance.
22. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
23. Implement the most common (quadratic) sorting algorithms.