C-ID Descriptor
Programming Concepts and Methodology II

Descriptor Details

- **Descriptor Title**: Programming Concepts and Methodology II
- **C-ID Number**: 132
- **Units**: 3.0
- **Date of Last Revision**: 10/12/2017 04:44:03 PM PDT

General Description

Application of software engineering techniques to the design and development of large programs; data abstraction and structures and associated algorithms.

Prerequisites

COMP 122

Corequisites

No information provided

Advisories

No information provided

Content

I. Programming Fundamentals (PF)
PF3. Fundamental data structures
Minimum coverage time: 12 hours

Topics

1. Primitive types
2. Arrays
3. Records
4. Strings and string processing
5. Data representation in memory
6. Static, stack, and heap allocation
7. Runtime storage management
8. Pointers and references
9. Linked structures
10. Implementation strategies for stacks, queues, and hash tables
11. Implementation strategies for trees
12. 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; and
8. Choose the appropriate data structure for modeling a given problem.

PF4. Recursion

Minimum coverage time: 5 hours

Topics

1. The concept of recursion
2. Recursive mathematical functions
3. Simple recursive procedures
4. Divide-and-conquer strategies
5. Recursive backtracking
6. 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; and
8. Determine when a recursive solution is appropriate for a problem.

II. Programming Languages (PL)
PL4. Declarations and types
Minimum coverage time: 3 hours

Topics
1. The conception of types as a set of values together with a set of operations
2. Declaration models (binding, visibility, scope, and lifetime)
3. Overview of type-checking
4. 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; and
6. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection).

PL5. Abstraction Mechanisms
Minimum coverage time: 3 hours

Topics
1. Procedures, functions, and iterators as abstraction mechanisms
2. Parameterization mechanisms (reference vs. value)
3. Activation records and storage management
4. Type parameters and parameterized types - templates or generics
5. 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; and
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
1. Object-oriented design
2. Encapsulation and information-hiding
3. Separation of behavior and implementation
4. Classes and subclasses
5. Inheritance (overriding, dynamic dispatch)
6. Polymorphism (subtype polymorphism vs. inheritance)
7. Class hierarchies
8. Collection classes and iteration protocols
9. 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; and
7. Describe how iterators access the elements of a container.

III. Software Engineering (SE)
SE1. Software design
Minimum coverage time: 8 hours

Topics
1. Fundamental design concepts and principles
2. Design strategy

Learning outcomes
1. Discuss the properties of good software design; and
2. Compare and contrast object-oriented analysis and design with structured analysis and design.

Lab Activities

No information provided

Objectives

At the conclusion of this course, the student should be able to:

1. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables
2. Implement, test, and debug simple recursive functions and procedures
3. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection)
4. Explain how abstraction mechanisms support the creation of reusable software components
5. Design, implement, test, and debug simple programs in an object-oriented programming language
6. Compare and contrast object-oriented analysis and design with structured analysis and design

Evaluation Methods

Exams
Quizzes
Programming Projects
Discussions
Class Presentations

Textbooks

Data Abstraction and Problem Solving with C++: Walls and Mirrors Latest Edition by Frank M. Carrano