Las Positas College
3000 Campus Hill Drive
Livermore, CA 94551-7650
(925) 424-1000
(925) 443-0742 (Fax)

Course Outline for CS 2

COMPUTING FUNDAMENTALS II

Effective: Spring 2016

I. CATALOG DESCRIPTION:
CS 2 — COMPUTING FUNDAMENTALS II — 4.00 units

Application of software engineering techniques to the design and development of large programs. Object-oriented programming methods and problem-solving strategies applied to intermediate-level problems using C++. Includes pointers and dynamic allocation; classes; encapsulation; inheritance and polymorphism; object and function overloading; recursive algorithms; data abstraction and structures.

3.00 Units Lecture 1.00 Units Lab

Strongly Recommended
CS 1 - Computing Fundamentals I

Grading Methods:
Letter or P/NP

Discipline:

	MIN
Lecture Hours:	54.00
Lab Hours:	54.00
Total Hours:	108.00

II. NUMBER OF TIMES COURSE MAY BE TAKEN FOR CREDIT: 1

III. PREREQUISITE AND/OR ADVISORY SKILLS:

Before entering this course, it is strongly recommended that the student should be able to:

A. CS1
1. Design, create and compile C++ programs within multiple development environments and operating systems, including the use of command-line tools in Unix/Linux.
2. Interpret and apply C++ control structures for sequencing, selection and iteration.
3. Interpret and implement programmer-defined functions in C++.
4. Create and interpret expressions involving arithmetic and logical operators;
5. Interpret and apply arrays and simple programmer-defined data structures in C++.
6. Modify and expand short programs that use standard conditional and iterative control structures and functions.
7. Choose appropriate conditional and iteration constructs for a given programming task.
8. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
9. Analyze and explain the behavior of simple programs.
10. Describe, interpret and apply the mechanics of parameter passing.
11. Discuss and apply the concept of algorithms in problem-solving processes.
12. Judge the correctness and quality of algorithms, identifying necessary properties of good algorithms.
13. Describe and apply effective debugging strategies.
14. Identify properties of variables and apply different forms of variable binding, visibility, scoping, and lifetime management.
15. Design, implement, test, and debug programs using basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.

IV. MEASURABLE OBJECTIVES:

Upon completion of this course, the student should be able to:

A. Write programs that use each a variety of data structures, such as arrays, records (structs), strings, linked lists, stacks, queues, and hash tables.
B. Implement, test, and debug simple recursive functions and procedures.
C. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection)
D. Explain how abstraction mechanisms support the creation of reusable software components.
E. Design, implement, test, and debug simple programs in C++.
F. Compare and contrast object-oriented analysis and design with structured analysis and design.
G. Describe how the class mechanism supports encapsulation and information hiding.
H. Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance.
I. Apply fundamental types and data structures, including pointers, to the creation of C++ programs.

J. Explain and apply C++ declarations and types.
K. Interpret and apply object-oriented programming concepts and syntax to the creation of C++ programs.

V. CONTENT:
A. Program Design and Development
1. Fundamental design concepts and principles
2. Design strategies
B. Fundamental Types and Data Structures
1. Primitive C++ types
2. Arrays
3. C++ structs (records)
4. Strings and string processing
5. Pointers
a. Definition and initialization
b. Pointer manipulation and dereferencing
c. Array name as a pointer
d. Dynamic allocation and runtime storage management
1. Comparison with garbage collection
6. References
a. Syntactic and semantic differences from pointers
7. Data representation in memory
8. Static, stack and heap allocation
9. Introduction to linked structures
a. Implementation strategies for stacks, queues and hash tables
b. Implementation strategies for trees
C. Declarations and types
1. Types as sets of values and sets of operations
2. Declaration models - binding, visibility, scope, lifetime
3. Overview of type-checking
D. Object-Oriented Programming
1. Encapsulation and information-hiding
2. Separation of behavior/interface and implementation
3. Member function design
a. Syntax and use
b. Const-ness
4. Constructors and Destructors
5. Access Levels and Scope
a. This pointer
b. Private vs protected vs public
6. Inheritance
a. Classes and subclasses
b. Overriding methods
c. Polymorphism
1. Dynamic dispatch
2. Virtual functions
d. Class hierarchies
7. Friend functions, methods and classes
8. Collection classes and iterators
9. Internal representation of objects, including method tables
E. Abstraction Mechanisms
1. Functions, methods and iterators as abstraction mechanisms
2. Parameterization mechanisms - pass-by-value vs pass-by-reference
3. Activation records and storage management
4. Parameterized types
a. Function templates
b. Class templates
5. Operator overloading
F. Introduction to Recursion
1. The concept of recursion
2. Recursive mathematical functions
3. Simple recursive procedures and divide-and-conquer strategies
4. Recursive backtracking

VI. METHODS OF INSTRUCTION:
A. **Classroom Activity** -
B. **Discussion** -
C. **Projects** -
D. **Lab** -
E. **Lecture** -
F. **Demonstration** -

VII. TYPICAL ASSIGNMENTS:

A. Write a C++ program that creates a DiceSet class representing a set of dice with a specifiable number and number of sides. Using objects of that class, implement a dice-based game (e.g., Pig, Yahtzee) playable by 2 or more players.
B. Write a C++ program that creates a Date class representing a calendar date, which implements operator overloading to allow objects of that class to be manipulated with arithmetic-like symbols such as "+" and "-" and to be used with standard stream output using the "<<" operator.
C. Write a C++ program that declares and implements a hierarchy of classes, using inheritance to organize common vs specialized features of four entities: Person, Employee, Manager, and Customer. Using objects from these classes, implement a program to manage creation, deletion and updating of relevant records for a small business.

VIII. EVALUATION:
A. **Methods**

1. Exams/Tests
2. Quizzes

3. Class Participation
4. Home Work
5. Lab Activities

B. **Frequency**

1. There should be at least two midterm examinations, or one midterm examination and several quizzes throughout the course.
2. One comprehensive final examination should be given at the end of the course.
3. Programming assignments (homework and lab activities) should cover each topic within the course content. Approximately one programming assignment per week is recommended.
4. Written homework assignments are recommended to parallel reading assignments and to be given weekly. However, it is optional to include written assignments in the instructor's official evaluation of students; instead, the instructor may choose to arrange written assignments to allow self-evaluation by students.
5. Class participation may optionally be included in student assessment. If so, it should be evaluated at least every other week, and encompass students' degree of engagement in the class relative to their peers.

IX. TYPICAL TEXTS:

1. Savitch, Walter. *Problem Solving with C++ .* 9th ed., Addison-Wesley, 2014.
2. Gaddis, Tony. *Starting Out with C++: From Control Structures through Objects .* 8th ed., Addison-Wesley, 2014.
3. Deitel, Paul, and Harvey Deitel. *C++ How to Program.* 9 ed., Prentice Hall, 2013.

X. OTHER MATERIALS REQUIRED OF STUDENTS:

A. It is strongly recommended that each student have a portable storage device (e.g, USB drive) and/or access to an individual account on a cloud-storage service.