Las Positas CurricUNET

Las Positas College

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

Course Outline for CS 20

ADVANCED PROGRAMMING WITH DATA STRUCTURES/C++

Effective: Fall 2018

I. CATALOG DESCRIPTION:
CS 20 — ADVANCED PROGRAMMING WITH DATA STRUCTURES/C++ — 4.00 units

Design and implementation of complex programs in C++ using a variety of fundamental data structures and algorithms. Includes the design and implementation of recursive functions, abstract data types, linked lists, stacks, queues, binary trees, hash tables, induction, searching and sorting algorithms, graphs, heaps and algorithm analysis.

3.00 Units Lecture 1.00 Units Lab

Strongly Recommended
CS 2 - Computing Fundamentals II
with a minimum grade of C

Grading Methods:
Letter or P/NP

Discipline:
- Computer Science

	MIN
Lecture Hours:	54.00
Expected Outside of Class Hours:	108.00
Lab Hours:	54.00
Total Hours:	216.00

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. CS2
1. Write programs that use each a variety of data structures, such as arrays, records (structs), 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 C++.
6. Describe how the class mechanism supports encapsulation and information hiding.
7. Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance.
8. Apply fundamental types and data structures, including pointers, to the creation of C++ programs.
9. Explain and apply C++ declarations and types.
10. Interpret and apply object-oriented programming concepts and syntax to the creation of C++ programs.

IV. MEASURABLE OBJECTIVES:

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

A. Apply and implement basic operations on abstract data types (ADTs)
B. Use pointers and dynamic memory allocation to create and manipulate complex data structures
C. Implement programs using linked lists, stacks, queues and binary trees
D. Design and implement larger programming projects with many inter-related components and classes
E. Explain and apply the concept of time efficiency for algorithms using Big O notation
F. Implement hash tables with multiple approaches to resolving collisions (e.g., linear probing, chaining)
G. Apply the technique of proof by induction to verify formulas for simple arithmetic series.
H. Solve a wide variety of problems using recursive functions and recursive programming techniques, including functional and procedural recursion, recursive functions with both arithmetic and logical outputs, and recursive backtracking.
I. Implement, apply and compare algorithms for searching and sorting.
J. Define graphs and implement standard algorithms operating on graphs.

V. CONTENT:

A. Data Abstraction
   1. Concept of an abstract data type (ADT)
   2. Typical operations in an ADT
   3. Implementating an ADT with multiple choices of internal data structure

B. Big O notation for the time complexity of algorithms
   1. Definition
   2. Calculation and simplification rules
   3. Application to standard searching and sorting algorithms

C. Linear structures
   1. Linked lists, singly and doubly linked
      a. Linked list applications
   2. Queues
      a. Implementation of queues by linked list and by circular array
   3. Stacks
      a. Implementation of stacks by array and by linked list
   4. Time complexity (Big-O) for traversal, insertion and deletion in the above data structures
   5. Applications such as evaluation of postfix (RPN) and infix expressions

D. Recursion
   1. Definition
   2. Techniques for building recursive functions
   3. Typical patterns of recursion
      a. Handle one vs handle the rest
      b. Divide and conquer
      c. Permutation
      d. Subsets
   4. Recursive backtracking

E. Search algorithms
   1. Direct access through use of hash codes
   2. Iterative binary search
   3. Recursive binary search
   4. Using a binary search tree
   5. Time complexity of search algorithms

F. Trees
   1. General definition, types, and terminology of trees
   2. Implementation of Binary Trees
   3. Applications of Binary Trees
   4. Complete Binary Trees: building a heap
   5. Time complexity of various Binary Tree operations

G. Sorting Algorithms
   1. Elementary iterative sorting algorithms (e.g., selection sort, insertion sort)
   2. Recursive sorting algorithms
      a. Merge sort
      b. Quick sort
      c. Heap sort
   3. Time complexity of sorting algorithms

H. Hash Tables
   1. Construction of an elementary hashcode function
   2. Resolving collisions by linear probing and chaining
   3. Applications of hash tables
   4. Cost tradeoff (memory vs time) of hash tables

I. Larger projects
   1. Design principles for classes and class hierarchies
   2. Top-down and bottom-up design
   3. Testing and debugging techniques

J. Proof by Induction
   1. Overall technique
   2. Application to arithmetic simple series

K. Graphs
   1. Definition of a graph
      a. Trees as a subcategory of graphs
   2. Searching and pathfinding within graphs
      a. Depth-first search
      b. Breadth-first search
      c. Djisktra's algorithm

VI. METHODS OF INSTRUCTION:

A. **Lab** -
B. **Projects** -
C. **Lecture** -
D. **Demonstration** -
E. **Discussion** -

VII. TYPICAL ASSIGNMENTS:

A. Write a C++ class that maintains the top 10 scores for a game application, implementing add and remove functions use a doubly linked list instead of an array
B. Create a C++ program that uses a hash table and related techniques to manage information about the "visit counts" for an arbitrary set of URLs
C. Write a program that can perform encryption and decryption using an arbitrary substitution cipher.
D. Implement the in-place heap-sort algorithm using c++. Compare its running time with that of the standard heap-sort that uses an external heap.
E. Write a class implementing a simple graph ADT that has only functions relevant to undirected graphs and does not include update functions using the adjacency matrix structure
F. Implement the B-tree data structure, assuming a block size of 1,024 and integer keys. Test the number of “disk transfers” needed to process a sequence of map operations

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 one midterm examination and one Final examination, preferably half way through the semester, and at the end of the semester.
2. There should be several quizzes throughout the course at appropriate times as new concepts are presented
3. 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.
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. Some larger assignments requiring multiple weeks and broken into sub-tasks, is recommended.
5. Programming lab assignments should cover each major topic within the course content. Approximately one programming assignment per week or per unit of instruction

IX. TYPICAL TEXTS:

1. *Data Structures and Algorithms in C++.* 4 ed., Cengage Learning, 2012.
2. *Programming Abstractions in C++.* 1 ed., Prentice Hall, 2013.
3. *Data Structures and Algorithm Analysis in C++.* 4 ed., Prentice Hall, 2013.
4. Dale, Nell. *C++ Plus Data Structures.* 6th ed., Jones & Bartlett Learning, 2016.
5. Malik, D. *C++ Programming Including Data Structures.* 8 ed., Course Tehnology, 2017.

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.