Lecture Contact Hours: 48-54; Homework Hours: 96-108; Total Student Learning Hours: 144-162
Laboratory Contact Hours: 48-54; Homework Hours: 0; Total Student Learning Hours: 48-54

CUYAMACA COLLEGE
COURSE OUTLINE OF RECORD

COMPUTER SCIENCE 281 – INTERMEDIATE C++ PROGRAMMING AND FUNDAMENTAL DATA STRUCTURES

3 hours lecture, 3 hours laboratory, 4 units

Catalog Description
Continuation of CS 181. Provides the programmer with professional training in memory management, documentation, structured programming, and programming to professional standards using C++. Explores some of the more advanced concepts of preprocessing, low-level data objects, recursion, and dynamic data structures including linked lists, stacks, queues and trees. Laboratory instruction includes program development and execution.

Prerequisite
"C" grade or higher or "Pass" in CS 181 or equivalent

Entrance Skills
Without the following skills, competencies and/or knowledge, students entering this course will be highly unlikely to succeed:
1) Write well-designed C++ code to solve a variety of scientific problems as standalone programs.
2) Select appropriate storage class characteristics for variables as well as methods.
3) Solve a variety of programming problems using sound structured, top-down, object-oriented design principles.
4) Use inheritance to minimize reinvention of similar objects.
5) Access standard C++ library modules.
6) Recognize the difference between (and implications of) reference vs. primitive data types.
7) Design, create and utilize complex classes and objects.
8) Input and output data to and from standard devices and files.
9) Visualize array subscripting as memory offset calculations and recognize some of the implications.
10) Understand and effectively utilize fundamental program control structures.
11) Understand the relative efficiency of the various sort and search algorithms.
12) Untangle poorly designed code.

Course Content
1) Pointers to variables, pointers to functions
2) Dynamic memory allocation/management
3) Class constructors, class destructors, class inheritance
4) Operator overloading, polymorphism
5) Linked lists, stacks, queues, trees
6) Recursion
7) Searching and sorting
8) Basic windows (MFC) programming: menus and toolbars

Course Objectives
Students will be able to:
1) Design and implement linked list memory management.
2) Design and implement a stacked or queued data structure.
3) Design and implement a tree data structure.

4) Given a binary tree, identify the order the nodes would be visited for preorder, inorder and postorder traversals.
5) Describe the behaviors and uses of hash tables, heaps and graphs.
6) Utilize a variety of searching and sorting algorithms; analyze their best, worst and average case performances.
7) Identify which dynamic memory data structures are appropriate for a variety of different applications based on algorithmic analyses.
8) Create code examples that demonstrate the ramifications of using references.
9) Explain the use of big-oh notation to describe the amount of work done by an algorithm.
10) Describe the strengths and drawbacks of both the store "by reference" and store "by copy" approaches to implementing data structures.

Method of Evaluation
A grading system will be established by the instructor and implemented uniformly. Grades will be based on demonstrated proficiency in subject matter determined by multiple measurements for evaluation, one of which must be essay exams, skills demonstration or, where appropriate, the symbol system.
1) Quizzes and exams that measure students' ability to use programming terminology and explain technical concepts related to data structures and their use in C++.
2) Practical exams requiring students to trace programming code to predict outcome. For example, a 10-20 line section of code is provided and students must be able to identify how all of the values in all variables are altered by that code and predict the final output as specified in the code.
3) Projects requiring students to write complete application programs demonstrating effective use of object-oriented principles, methodology and data structures.

Special Materials Required of Student
Flash drive

Minimum Instructional Facilities
Computer lab with networked computers and software to create C++ programming environment

Method of Instruction
1) Lecture and demonstration
2) Hands-on lecture
3) Lab problems

Out-of-Class Assignments
1) Design, code and debug multiple C++ programs and objects that demonstrate class objectives
2) Analyze instructor-assigned pre-coded C++ programs and post analysis comments on the class discussion board

Texts and References
1) Required (representative example): Think C++ by Allen Downey, Green Tea Press; PDF; 2020.
2) Supplemental: None

Exit Skills
Students having successfully completed this course exit with the following skills, competencies and/or knowledge:
1) Write, test and debug recursive methods.
2) Utilize inheritance and polymorphism to minimize reinvention of code.
3) Design and create a program using standard linked list memory management.
4) Design and create a program implementing a stack or queue data structure.
5) Design and create a program implementing a tree data structure.
6) Understand the behaviors and uses of hash tables, heaps and graphs.

7) Utilize a variety of searching and sorting algorithms; analyze their best, worst and average case performances.
8) Identify which dynamic memory data structures are appropriate for a variety of different applications based on algorithmic analyses.
9) Write larger, more complex programs.
10) Develop sound software engineering skills.

Student Learning Outcomes

Upon successful completion of this course and given a scientific problem-based scenario, students will be able to:

1) Decompose problems and design program solutions using flowcharts, pseudocode, models, or other tools.
2) Properly code applications using the fundamental coding structures: sequence, selection, and loops.
3) Test and debug applications using debugging tools such as trace execution.