Mt. San Jacinto College
Integrated Course Outline of Record

Form B

Submitted by: Glenn Stevenson **Date:** 12/17/2019

Department	Subject	Course Number	Title
Computer Sci/ Info Systems	Computer Sci/ Info Systems CSIS	211	Introduction to Data Structures and Algorithms

Units/Hours

Each lecture unit requires 1 hour per week of class time, and 2 hours per week of study outside of class.
Each laboratory unit requires 3 hours per week of class time.
Each activity unit requires 2 hours per week of class time and 1 hour per week of study outside of class.

Lecture Units	Total Units
3.00	3.00
Lecture Contact Hours	**Total Contact Hours**
48.00 - 54.00	48.00 - 54.00
Lecture Homework Hours	
96.00 - 108.00	

Stand Alone:

Program Applicable

AA/AS Degree General Ed Breadth Area(s):

-none-

General Education Justification:

Maximum Enrollment: 30

Maximum Enrollment Justification:

Course requires significant response to written materials - check all that apply:

* Course requires more than seven 3+ page papers to grade per student per semester.
* Course requires an unusually large amount of written work to be responded to individually by the instructor per semester.

Course requires significant individualized instruction or

assessment â€“ check all that apply:

*** Course requires that each student be evaluated individually on a set of skills more than twice per semester.**

Justification: This is an advanced course. Students will have to create at least 8 projects that will contain up to 8 source code files. Coding is not like English. Source code files can be extremely large. In addition, students are solving extremely complex logic problems that have to evaluated. Grading these projects also involves a lot of written analysis of how the project was structured, how well it was coded and how efficient the coded project was solved.

Grading Method: Letter Grade or P/NP

TOP code: 0706.00

Can be Taken 1 **time(s) for credit** (max 4)

- Visual or Performing Arts course that is required to meet major requirements for UC/CSU
- Intercollegiate athletics course
- Academic/vocational competition course

Catalog Description:

(Please do not refer to transferability or degree, certificate, or employment concentration applicability. Please only describe the course).
(**75 words** or less in gray box below).

This course is intended to introduce students to the concept of data structures and algorithms. Basic topics in this course include arrays, lists, stacks and queues. Advanced topics such as dictionaries including binary search trees, hashing, priority queues, and heaps will also be covered. In addition, this course will introduce analysis of algorithms, sorting algorithms, and object-oriented programming techniques including abstract data types, inheritance, and polymorphism.

Schedule Description:

(Please do not refer to transferability or degree, certificate, or employment concentration applicability. Please only describe the course).
(**25 words** or less in gray box below).

This course is an advanced programming course that teaches the fundamentals of data structures and algorithms using the C++ and Java programming languages.

Need for the course:

This course is a requirement for the AS-T degree in Computer Science. It is is also an elective course for the AS Degree in Computer Programming.

Prerequisite(s):

Prerequisites go through a separate approval process. See Forms E1-E6 for details.
(For further clarification, contact the Prerequisite Subcommittee)

- CSIS 113A with a Grade of C or better. or
- CSIS 113B with a Grade of C or better.

Corequisite(s):

Corequisites go through a separate approval process. See Forms E1-E6 for details.

-none-

Recommend Preparation:

Recommended Preparation goes through a separate approval process. See Forms E1-E6 for details.

-none-

Other Enrollment Criteria:

-none-

Learning Objectives:

(please number each objective and express in behavioral terms)

Upon the completion of the course the student will be able to do the following:

1. Compose computer programs using a high level programming languages as a means of problem solving. This includes abstract data types, inheritance, and polymorphism.
2. Evaluate the advanced features of an object oriented programming language including arrays, pointer manipulation, recursion, and object oriented programming.
3. Discover how data is stored and manipulated in a computer by applying searching and sorting alogorithms.
4. Analyze algorithm complexity using Big O notation.
5. Develop a fundamental knowledge of the practical uses of data structures and their applications. This includes stacks, queues, lists, and binary trees
6. Evaluate sorting and searching algorithms complexities and their usage
7. Solve sorting and searching problems by applying the most efficient algorithm.

Course Content:

(please number the outline of main topics and subtopics)

1. Object Oriented programming
    1. Problem Solving.
    2. Object Oriented Design.
    3. Implementing Software Design.
    4. Testing classes.
    5. Implementing and testing algorithms.
    6. UML.
    7. Inheritance aggregation and composition.
    8. Mutable and immutable objects.
2. Abstract Data Types
    1. Concrete Data Types.
    2. Abstraction.
    3. Preconditions and Post conditions.
    4. Using Abstract data types in an algorithm.
    5. Concreted data types.
    6. Using the assert statement.
    7. Class invariants.
    8. Inheritance and polymorphism.
3. Arrays
    1. Simple array algorithms.
    2. Arrays of objects.
    3. Sequential search.
    4. Complexity Analysis.

1. Big O Notation.
5. Binary Search.
6. Vectors.
7. Multidimensional arrays.

4. Linked Structures
    1. Maintaining an ordered array.
    2. Indirect reference.
    3. Linked nodes.
    4. Inserting nodes into a linked list.
    5. Inserting at the front of a list.
    6. Deleting from a sorted linked list.
    7. Nested classes.
5. Stacks
    1. The stack abstract data type.
    2. An array implementation of a stack.
    3. Evaluating post fix notation.
    4. Linked implementation.
6. Queues
    1. The queue abstract data type.
    2. A linked implementation of a queue.
    3. Simulation with queues.
    4. Circular queues.
7. Collections
    1. Iterators
    2. Abstract collection class.
    3. Array collection class.
    4. Linked collection class.
8. Lists
    1. Bidirectional list iterators.
    2. Lists and polynomial algebra.
9. Hash tables
    1. Tables and records.
    2. An abstract data type for maps.
    3. Linear probing.
    4. Rehashing.
    5. Collision resolution algorithms.
    6. Separate chaining.
    7. Analysis of hash functions.
    8. Perfect hash functions.
    9. Other hash functions.
10. Recursion
    1. Recursive functions.
    2. The Towers of Hanoi.
    3. Fibonacci numbers.
    4. Calling trees.
    5. Storing instead of recomputing.
    6. De Moivre’s formula.
    7. Recursive binary search algorithms.
    8. Recursive exponentiation.
    9. Printing permutations.
    10. Indirect recursion.
11. Trees
    1. Properties of trees.
    2. Recursive definition of trees.

3. Decision trees.
4. Ordered trees.
5. Traversal algorithms for ordered trees.
6. Complete ordered trees.

12. Binary Trees
    1. Properties of binary trees.
    2. Counting binary trees.
    3. Binary tree traversal algorithms.
    4. Expression trees.
    5. Forests.
13. Search Trees
    1. Keys and comparable types.
    2. Binary search tree deletion.
    3. Binary search tree performance.
    4. AVL trees.
    5. Implementing AVL trees.
    6. Fibonacci trees.
    7. Multi-way search trees.
    8. B-Trees.
    9. Red-black trees.
14. Heaps and Priority Queues
    1. Heaps.
    2. Heap algorithms.
    3. Priority Queues.
    4. Huffman codes.
15. Sorting
    1. Bubble sort.
    2. Selection sort.
    3. Insertion sort.
    4. Shell sort.
    5. Speed limit for comparison sorts.
    6. Merge sort.
    7. Quick sort.
    8. Heap sort.
    9. Bucket sort.
    10. Radix sort.
16. Graphs
    1. Adjacency matrix implementation.
    2. Adjacency list implementation.
    3. Breadth-first search.
    4. Spanning trees.
    5. Depth-first search.
    6. Weighted graphs.
    7. Dijkstra's algorithm.
    8. Digraphs.

Methods of Instruction:

Methods of instruction may include, but are not limited to the following:

- **Method:** Lecture
  Integration: Lecture, with supporting visual materials (PowerPoint presentation or multimedia), will introduce conceptual and practical skills throughout the course, such as, using a high level programming languages as a means of problem solving, evaluating the advanced features of an object oriented programming language, analyzing algorithm complexity using Big O notation, the practical uses of data structures and their applications

Algorithms.
- **Method:** Discussion
  Integration: Weekly discussion will be conducted as a means to analyze algorithm complexity, discuss the practical applications of data structures, discover how data is stored and manipulated by applying searching and sorting algorithms, and determine which searching and sorting algorithms are most efficient for a given application.
- **Method:** Observation and Demonstration
  Integration: Observation and demonstration will illustrate lecture principles and reading assignments, with a focus on high level programming language concepts and applying them to the Data Structures and algorithms required for homework assignments. Guided practice will include programs to solve sorting and searching problems efficiently, the creation of stacks, queues, lists, and binary trees, solving complex problems with recursion, and the proper use of pointer manipulation and polymorphism.

Methods of Evaluation:

A student's grade shall be determined by the instructor using multiple measures of performance related to the course objectives. Methods of evaluation may include but are not limited to the following:

- **Method:** Exams/Tests
  Integration: A midterm and a final exam composed of short answer questions and small programming problem will be given. The midterm will be designed to show a studentâ€™s understanding of algorithm analysis in terms of Big O notation and the application of Data Structures. The final exam will be designed to show the students understanding of data structures concepts such as applying the most efficient algorithm to solve searching and sorting problems and the practical uses of stacks, queues, lists, and binary trees.
- **Method:** Homework
  Integration: Several large scale programming assignments consisting of a minimum of 8 source code files will be required. The projects will consist of complex logic problems that are created from the weekly lecture material and are intended to demonstrate the students understanding of the practical application of stacks, queues, lists, and binary trees. Assignments will also be geared to evaluate the students understanding of algorithm complexity and searching and sorting algorithms.
- **Method:** Guided Practice
  Integration: Guided practice consisting of complex Data Structures problems and algorithm analysis will be given to show a studentâ€™s understanding of algorithm complexity, binary trees, sorting methods and Big O notation.

Examples of Assignments:

Students will be expected to understand and critique college level texts or the equivalent. Reading and writing, as well as out of class assignments are required. These assignments may include but are not limited to the following:

Assignment 1

Create a linked binary tree that will store 100 numbers between 1 and 100. Using one of the popular search algorithms discussed in lecture, produce a driver that will search for a number within your tree. You should ask the user to input a number then go find it. Keep count as to how many branches of the tree you have traversed and tell the user where you found it. If the number is not found then print out that it hasn't been found.

Assignment 2

Create A Stack class written in C++ using templates or Java using Generics. Your class should allow for a stack of any data type and should include the following functionality:
1. A push function that adds data to the top of the stack.
2. A pop function that removes and returns data from the top of the stack.

3. A peek function that will return the value at the top of the stack without removing it.

Textbooks:

- Cutajar, James (2018). *Beginning Java Data Structures and Algorithms* Packt Publishing. ISBN: 9781789537178

Other Resources:

Minimum Qualification

- Computer Science (Masters Required)