CS 113 - Basic Data Structures and Algorithms

Units Lecture 2.00	**Units Lab** 1.00	**Units Total** 0.00 - 3.00
Lecture Weekly Contact Hours 2.00	**Lab Weekly Contact Hours** 3.00	**Total Weekly Contact Hours** 5.00
Lecture Weekly Out of Class Hours 4.00	**Lab Weekly Outside of Class Hours** 0.00	**Total Weekly Outside of Class Hours** 4.00
Total Contact Hours 80.00 - 90.00	**Total Outside of Class Hours** 64.00 - - 72.00	**Total Course Hours** 144.00 - 162.00

Typically Offered: Fall, Spring, and Summer - F,SP,SU

COURSE DESCRIPTION

The course uses topics of personal and social relevance to investigate the impact of computing through efficient algorithms and properly designed data structures. Students explore the software development process by developing effective solutions using industry-standard tools. Topics include searching, sorting, hashing, algorithm analysis, object-oriented design, collections, lists, stacks, queues, trees, sets, dictionaries, and graphs. C-ID COMP-132.

ENROLLMENT RESTRICTIONS

Prerequisite
CS 112

OUTLINE OF COURSE LECTURE CONTENT

The course lecture will address the following topics:
I. Object-oriented programming (OOP) design and algorithmic analysis
A. Computationally efficient coding through the use of appropriate design tools such as algorithms, pseudo-code, Unified Modeling Language (UML), and flow diagrams before coding
B. Growth functions and "Big O" notation
C. Growth function comparison as the number of inputs increases
D. Time complexity issues
E. Testing
1. Preparations for testing
2. Developing appropriate test data
3. Testing boundary conditions.

II. Data structures and collections
A. Stacks, heaps, queues, priority queues, and lists
B. Linked lists, trees, binary trees, binary search trees, 2-3-4 trees, and skip lists
C. Application and implementation approaches.

III. Graphs, hash tables, sets, and maps
A. Networks and graphing strategies
B. Hashing functions and resolving collisions
C. Implementation strategies.

IV. Recursion
A. Divide and conquer technique
B. Dynamic programming.

V. Analysis and efficiency of sort and search methods
A. Selection, insertion, shellsort, and bubble sorts
B. Quicksort, merging and mergesort, heapsort, and radix sorts
C. Performance characteristics of search routines.

OUTLINE OF COURSE LAB CONTENT

The course lab will address the following topics:
Individual and group projects in the lab are designed to be hands-on activities that support, compliment, and extend the material and theory presented in the lectures. Students gain significant project experience.

PERFORMANCE OBJECTIVES

Upon successful completion of this course, students will be able to do the following:
1). Perform a thorough analysis of a specified problem and then carefully design a solution using appropriate design tools, such as algorithms, Unified Modeling Language (UML), and flow diagrams.
2). Work with Java's Collections Framework and assess the advantages and disadvantages of various data structures in terms of efficiency, specific program appropriateness, and implementation considerations.
3). Utilize built-in collection classes, such as the ArrayList, LinkedList, Stack, Queue, and Tree classes, to store, process, and retrieve data efficiently.
4). Use "Big O" notation to analyze the efficiencies of various data structures relative to their growth rates as the data set being used is increased in size.
5). Implement various sort and search algorithms in working programs and use "Big O" notation to analyze their efficiency.
6). Thoroughly test a program's accuracy as well as its usefulness for the intended problem.
7). Use industry-standard development tools for version control, unit testing, and continuous integration.
8). Explain how computer science relates to the students' lived experiences and their future in it.

READING ASSIGNMENTS

Reading assignments will be consistent with, but not limited by, the following types and examples:
1). Read about object-oriented programming using the Java language in the class text as well as using and understanding specialized terms, concepts, and theories relative to data structures and algorithms.
2). Re-enforce text topics by researching the material on the Web and other texts for further explanation.

3). Read instructor-distributed handouts on specialized topics relevant to successful programming in industry using Java and extending this knowledge to class projects.

WRITING ASSIGNMENTS

Writing assignments will be consistent with, but not limited by, the following types and examples:
1). Create and write a requirements specification that defines what the program will accomplish and how the user will interact with it for each project done as homework and in class.
2). Write a report on the criteria that should be addressed by the programmer before selecting a data structure for use in a program and be prepared to debate the topic with classmates.
3). Write a report on the efficacy and efficiency of the various sort and search methods discussed in class and in the textbook.

OUTSIDE-OF-CLASS ASSIGNMENTS

Outside-of-class assignments will be consistent with, but not limited by, the following types and examples:
1). Complete weekly programming assignments including a requirements specification.
2). Complete individual and group programming projects.
3). Prepare for individual presentations on specific topics discussed in the text.

STUDENT LEARNING OUTCOMES

1. The student will be able to analyze abstract programming problems involving data sets and produce software designs that utilize appropriate data structures. This includes analyzing the efficiency of an algorithm in comparison with other similar algorithms.

2. The student will generate solutions to programming problems using existing data structures as well as employ efficient and appropriate algorithms to access and process the data contained within the data structures

3. The student will be able to describe basic data structures and explain the details of common algorithms used to manipulate the data contained in those data structures both verbally and in writing.

4. The student will be able to work in a team that will agree on an organizational model, establish roles and delegate tasks to develop designs which incorporate data structures and algorithms and produce a programming project in a timely, cooperative, and collective manner.

METHODS OF INSTRUCTION

Instructional methodologies will be consistent with, but not limited by, the following types or examples:
1). Instructor lecture and guided discussion.

2). In-class quizzes and lab assignments to give student programmers the opportunity to practice critical thinking skills as they relate to the efficacy and efficiency of their programs.

3). Group projects to provide student programmers an opportunity to learn from each other as well as get valuable feedback on their programming skills and ability to work in a team environment simulating industry conditions.

4). Policies and practices that promote ownership of active learning, such as the following:

- Late policy that allows flexibility for various student circumstances.
- Opportunities to resubmit assignments to demonstrate continuous improvement.
- Personalizing choices of assessments to learn CS and apply it to their life.
- Regular interactions with students to provide support and feedback.

METHODS OF EVALUATION

Evaluation methodologies will be consistent with, but not limited by, the following types or examples:
1). Discussions evaluated for the student's demonstrated ability to relate computer science to their life.

2). In-class activities evaluated for the student's ability to explain and use data structure development and selection.

3). Participation in campus events evaluated for the student's ability to relate computer science to their lived experiences.

4). Programming labs and exercises evaluated for demonstrated improvement in the student's ability to debug and resolve errors that occur while programming.

5). Unit deliverables evaluated for the student's ability to design and implement user-centered solutions that relate to student's life and interests.

REQUIRED TEXTBOOKS

Examples of typical textbooks for this course include the following:

1. **Author** Koffman, Elliot B., and Paul A. T. Wolfgang
   Title Data Structures: Abstraction and Design Using Java
   Edition 4th ed.
   Publisher Wiley
   Year 2021
   ISBN 978-1119703617
   This is the most current, in-print edition. No

COURSE REPEATABILITY

Total Completions Allowed: 1

Rationale for multiple enrollments:

Courses Related in Content (CRC) in Physical Education, Visual Arts, and Performing Arts:

DISTANCE ED (FORM A)

Type of Approval: 100% Online or Hybrid

You may indicate here which component(s) of the course should never be conducted online (e.g. proctored exams, labs, in-person orientation, etc.):

ARTICULATION

Transfer Status: Acceptable for Credit: CSU, UC -

CSU/IGETC GE Area(s): 103 - CSU, UC

THIS COURSE IS INCORPORATED INTO THE FOLLOWING PROGRAM(S)

Computer Science for Transfer *CURRENT* AS-T Degree
Software Development *CURRENT* AA Degree
Software Development *CURRENT* Certificate of Achievement
Software Development *FUTURE* Certificate of Achievement
Software Development *FUTURE* AA Degree