Catalyst Information

Dept and Nbr: CS 10C        Title: PROGRAMMING CONCEPTS 2
Full Title: Programming Concepts and Methodologies 2
Last Reviewed: 5/8/2017

Catalog Description:
Application of software engineering techniques to the design and development of large programs; data abstraction and structures and associated algorithms.

Prerequisites/Corequisites:
Course Completion of CS 10B

Recommended Preparation:
Eligibility for ENGL 1A or equivalent

Limits on Enrollment:

Schedule of Classes Information:
Description: Application of software engineering techniques to the design and development of large programs; data abstraction and structures and associated algorithms. (Grade or P/NP)
Prerequisites/Corequisites: Course Completion of CS 10B
Recommended: Eligibility for ENGL 1A or equivalent
Limits on Enrollment:
Transfer Credit: CSU;UC.
Repeatability: Two Repeats if Grade was D, F, NC, or NP
ARTICULATION, MAJOR, and CERTIFICATION INFORMATION:

AS Degree: Area
Effective: Inactive:
CSU GE: Transfer Area
Effective: Inactive:
IGETC: Transfer Area
Effective: Inactive:

CSU Transfer: Transferable Effective: Spring 1991 Inactive:
UC Transfer: Transferable Effective: Spring 1991 Inactive:

CID:
CID Descriptor: COMP 132 Programming Concepts and Methodology II
SRJC Equivalent Course(s): CS10C

Certificate/Major Applicable:
Major Applicable Course

COURSE CONTENT

Student Learning Outcomes:
Upon completion of the course, students will be able to:
1. Write programs in C++ that use arrays, linked lists, stacks, queues, hash tables, and recursion.
2. Explain how object-oriented programming uses abstraction to increase reusability of software.
3. Summarize the differences between programming paradigms.

Objectives:
Upon completion of the course, students will be able to:
1. Write programs that use each of the following data structures: arrays, records, 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 an object-oriented programming language.
6. Compare and contrast object-oriented analysis and design with structured analysis and design.

Topics and Scope:

I. Programming Fundamentals
   A. Primitive types
   B. Arrays
   C. Records
   D. Strings and string processing
   E. Data representation in memory
   F. Static, stack, and heap allocation
   G. Runtime storage management
   H. Pointers and references
I. Linked structures
   J. Implementation strategies for stacks, queues, and hash tables
   K. Implementation strategies for trees
   L. Strategies for choosing the right data structure

II. Recursion
   A. The concept of recursion
   B. Recursive mathematical functions
   C. Simple recursive procedures
   D. Divide-and-conquer strategies
   E. Recursive backtracking
   F. Implementation of recursion

III. Declarations and Types
   A. The conception of types as a set of values together with a set of operations
   B. Declaration models (binding, visibility, scope, and lifetime)
   C. Overview of type-checking
   D. Garbage collection

IV. Abstraction Mechanisms
   A. Procedures, functions, and iterators as abstraction mechanisms
   B. Parameterization mechanisms (reference vs. value)
   C. Activation records and storage management
   D. Type parameters and parameterized types - templates or generics
   E. Modules in programming languages

V. Object-Oriented Programming
   A. Object-oriented design
   B. Encapsulation and information-hiding
   C. Separation of behavior and implementation
   D. Classes and subclasses
   E. Inheritance (overriding, dynamic dispatch)
   F. Polymorphism (subtype polymorphism vs. inheritance)
   G. Class hierarchies
   H. Collection classes and iteration protocols
   I. Internal representations of objects and method tables

VI. Software Design
   A. Fundamental design concepts and principles
   B. Design strategy

All topics are covered in both the lecture and lab parts of the course.

Assignment:

Lecture Related Assignments:
1. Read approximately 30 pages per week
2. Complete 2-8 examinations including final exam

Lab Related Assignments:
1. Complete 10-15 programming assignments, with documentation, using the C++ programming language

Methods of Evaluation/Basis of Grade:
**Writing:** Assessment tools that demonstrate writing skills and/or require students to select, organize and explain ideas in writing.

| Written program documentation | 10 - 20% |

**Problem Solving:** Assessment tools, other than exams, that demonstrate competence in computational or non-computational problem solving skills.

| Programming assignments | 20 - 60% |

**Skill Demonstrations:** All skill-based and physical demonstrations used for assessment purposes including skill performance exams.

| None | 0 - 0% |

**Exams:** All forms of formal testing, other than skill performance exams.

| Exams, Final Exam: (Multiple choice, true/false, matching items, completion, programming problems) | 20 - 60% |

**Other:** Includes any assessment tools that do not logically fit into the above categories.

| None | 0 - 0% |

**Representative Textbooks and Materials:**
Starting Out with C++ From Control Structures through Objects. 8th ed. Gaddis, Tony. Pearson. 2014