CS A122: PROGRAMMING CONCEPTS AND METHODOLOGY 1

Item	Value
Curriculum Committee Approval Date	11/02/2016
Top Code	070600 - Computer Science (Transfer)
Units	3 Total Units
Hours	90 Total Hours (Lecture Hours 36; Lab Hours 54)
Total Outside of Class Hours	0
Course Credit Status	Credit: Degree Applicable (D)
Material Fee	No
Basic Skills	Not Basic Skills (N)
Repeatable	No
Grading Policy	Standard Letter (S)

Course Description

Introduction to the discipline of Computer Science using a high-level language utilizing programming and practical hands-on problem solving. The first course for students seeking the Computer Science AS-T transfer degree. Advisory: Computer Information Systems A090 or A100 or A111. Transfer Credit: CSU, UC. C-ID COMP 122

Course Level Student Learning Outcome(s)

1. Students will implement different programs using each of the following: basic computation, simple I/O, conditional and iterative structures.
2. Students will write programs that make use of function definitions, decomposition, parameter passing,and arrays.

Course Objectives

- I Programming Fundamentals
- I. 1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs.
- I. 2. Modify and expand short programs that use standard conditional and iterative control structures and functions.
- I. 3. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.
- I. 4. Choose appropriate conditional and iteration constructs for a given programming task.
- I. 5. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
- I. 6. Describe the mechanics of parameter passing.
- II Algorithms and Problem Solving
- II. 1. Discuss the importance of algorithms in the problem-solving process.
- II. 2. Identify the necessary properties of good algorithms.
- II. 3. Create algorithms for solving simple problems.
- II. 4. Use pseudocode or a programming language to implement, test, and debug algorithms for solving simple problems.
- II. 5. Describe strategies that are useful in debugging.
- III Overview of Programming Languages
- III. 1. Summarize the evolution of programming languages, illustrating how this history has lead to the paradigms available today.
- III. 2. Identify at least one distinguishing characteristic for Object-Oriented and Procedural programming.
- IV Declarations and Types
- IV. 1. Explain the value of declaration models, especially with respect to programming-in-the-large.
- IV. 2. Identify and describe the properties of a variable, such as its associated address, value, scope, persistence, and size.
- IV. 3. Discuss type incompatibility.
- IV. 4. Demonstrate different forms of binding, visibility, scoping and lifetime management.
- IV. 5. Defend the importance of types and type-checking in providing abstraction and safety.

Lecture Content

Introduction to Programming Computers and Memory: hardware and how data is stored in memory Software and the CPU: types of software; how the CPU works Programming Mechanics: using a development environment Syntax Basics: analyzing your first program Problem Solving: Algorithm Design Data and Simple Output History of Programming: machine, assembly and high-level languages Modern Programming Languages and Paradigms Console output with strings and escape sequences Variables and Values: how to declare variables and fill them with values Numbers: introduction to the integer and real number types Objects, Input and Processing Interactive Programs: create programs that accept user input Processing Data: use arithmetic operators to perform calculations Math Functions: learn how to use library functions Declarations and data types: static and dynamic typing, casting Characters and Strings, parsing and formatted output Problem Solving: Hand-tracing algorithms before coding Introducing Computer Logic and Decisions True False: learn how to represent Boolean values Decision Relationships: comparing numbers, objects and strings Introducing Selection: make decisions with the if, else and conditional operators Logical Operators: combine Boolean expressions using logical operators Multiple Choice: make decisions involving multiple inputs and outputs Problem Solving: Flowcharts and Test Cases Application: Input Validation Loops and Repetition Writing Loops: learn a strategy for writing loops that are correct. Counter-controlled loops: learn how to use loops that rely on a counter Sentinel and flag-controlled loops: learn how to wr ite indefinite loops Using do-while, managing necessary and intentional bounds Jump statements: using break and continue to fine-tune a loop Iterators controlled loops: use iterators to process Files and collections Limit-controlled loops: write loops that use a limit bound Procedures and Functions Procedural Programming; history and concepts of structured programming Writing Procedures: learn the syntax of procedures Passing Parameters: write procedures that modify behavior based on passed arguments Writing Functions: learn to write methods that calculate and return a value Syntax Errors: learn techniques for avoiding common compiler errors Problem Solving: writing reusable methods Problem Solving: functional decomposition and stepwise refinement Variable scope and variable lifetime (automatic variables and the stack) Introducing Array and Lists Lists and Arrays: how to create and initialize one-dimensional primitive arrays Lists and Methods: learn how arrays are stored and how to write methods to process them Lists and Loops: use loops to implement common array-processing algorithms List Algorithms: extreme values, sum/average/count, adjacent elements Matrices: create lists that contain rows and columns Bugs and Testing Runtime Errors: learn to recognize runtime errors, throw exceptions and use assertions Unit Tests: learn how to create and use unit tests (Manual and with a framework) Debugging: learn how to use your IDEs debugger and a debugging strategy Problem Solving: Tracing Objects Files and Exceptions Reading and Writing Text files Text Input and Output Command Line Arguments Exception Handling Application: Handling I nput Errors

Lab Content

Use an IDE or other tools to compile, run and test programs. Examine and understand the way that computer memory is arranged and how different types of data are stored. Create and test programs that: Perform interactive Input/Processing/Output Process strings with loops or iteration Perform decision making with selection Process arrays or lists including common summation and extreme value algorithms. Perform input and output on files included formatted output.

Method(s) of Instruction

- Lecture (02)
- DE Online Lecture (02X)
- Lab (04)
- DE Online Lab (04X)

Instructional Techniques

Lecture, demonstration and programming exercises.

Reading Assignments

Students will spend a minimum of 4 hours per week reading the textbook and/or other reading material assigned. Students will be expected to follow along with the exercises in the reading material.

Writing Assignments

Students will spend a minimum of 6 hours per week writing code.

Out-of-class Assignments

Students will spend a minimum of 6 hours per week completing weekly programming assignments.

Demonstration of Critical Thinking

Students will demonstrate the ability to write programs that solve different kinds of problems.

Required Writing, Problem Solving, Skills Demonstration

Students will demonstrate proficiency writing computer programs.

Eligible Disciplines

Computer science: Masters degree in computer science or computer engineering OR bachelors degree in either of the above AND masters degree in mathematics, cybernetics, business administration, accounting or engineering OR bachelors degree in engineering AND masters degree in cybernetics, engineering mathematics, or business administration OR bachelors degree in mathematics AND masters degree in cybernetics, engineering mathematics, or business administration OR bachelors degree in any of the above AND a masters degree in information science, computer information systems, or information systems OR the equivalent. Note: Courses in the use of computer programs for application to a particular discipline may be classified, for the minimum qualification purposes, under the discipline of the application. Masters degree required.

Textbooks Resources

1. Required Roberts, E.. Understanding Programming through JavaScript, 1 ed. Hoboken, NJ: Pearson, 2020