CSCI 0046 - SYSTEM PROGRAMMING WITH C

Catalog Description

Prerequisite: Completion of CSCI 12 with grade of "C" or better
Advisory: Completion of CSCI 50 with grade of "C" or better
Hours: 72 (54 lecture, 18 laboratory)
Description: Introduction to the C language and system programming on a Unix-like operating system. Topics include the standard C library, memory allocation, file I/O, permissions, system calls, networking, and process management. Development in a Unix environment will cover editors, shell scripting, makefiles, source code control, and debugging. (CSU, UC)

Course Student Learning Outcomes

- CSLO #1: Use the standard C libraries for input/output, memory management, and networking.
- CSLO #2: Construct multi-file programs using appropriate software engineering tools.
- CSLO #3: Construct programs utilizing arrays, structures, loops, and/or subroutines.
- CSLO #4: Debug and add features to existing programs.

Effective Term

Fall 2020

Course Type

Credit - Degree-applicable

Contact Hours

72

Outside of Class Hours

90

Total Student Learning Hours

162

Course Objectives

Lecture Objectives:
1. Identify the phases of compiling a C program.
2. Interpret a written program specification and decompose it into psuedocode.
3. Analyze the dependency graph for building a multi-file project. Construct a Makefile to automate the build process.
4. Describe the primitive C data types.
5. Interpret documentation of library functions and use the functions in working programs.
Laboratory Objectives:
Interpret written program specifications and write C programs conforming to the ANSI standard incorporating the following language and library features:
1. Standard I/O using printf, scanf, fputs, and fgets;
2. String manipulation using strcmp, arrays, and pointer arithmetic;
3. Structs and unions;
4. Multiple source and header files;
5. Stream-based and record-based file I/O;
6. Dynamic memory management;
7. Process management system calls including fork, exec, and wait, and their variants;
8. Command-line arguments;
9. Control structures: if/else, while, do/while, for;
10. Functions that accept and return primitive types, arrays, and structs;
11. Standard mathematical functions from the libm library;
12. Client socket programming;
13. Recursive functions.
14. Writing makefiles to automate the compilation process;
15. Coordinating development among two or more persons using a version control system;
16. Compiling and linking programs requiring multiple source files.

General Education Information

- Approved College Associate Degree GE Applicability
- CSU GE Applicability (Recommended-requires CSU approval)
- Cal-GETC Applicability (Recommended - Requires External Approval)
- IGETC Applicability (Recommended-requires CSU/UC approval)

Articulation Information

- CSU Transferable
- UC Transferable

Methods of Evaluation

- Classroom Discussions
  - Example: Why is fgets preferred over gets? Answer: The gets function doesn't do bounds-checking on the string array. It is possible to overrun the array by simply inputting more characters than the array allows for. The fgets functions, by contrast, is passed the length of the array and will not accept more characters than the given limit. Buffer overruns and a myriad of security vulnerabilities can be prevented by using fgets instead of gets. Grade based on participation.
- Objective Examinations
  - Example: Sample test question: The following statements are executed: int x, y; x = 5; y = 8; y = ++x * y; What are the values of x and y? Answer: x = 6, y = 40
- Problem Solving Examinations
  - Example: Sample assignment: Write a recursive function to implement the quicksort algorithm. Write a driver program to accept a list of numbers from the user, sort them, and display the results to standard output. Test your function on integer arrays of size zero, one, two, and ten, and sixteen. Rubric Grading: - Does the program accept numeric input and display correct results? - Is the function recursive? - Does the function work on inputs of size zero, one, two, ten, and sixteen? - Does the function choose a suitable pivot element? - Is the relative ordering of elements retained as they are copied around the pivot? - Is the program well-documented? - Does the program include a Makefile? - Does the program compile without warnings or errors?
- Projects
  - Example: Sample assignment: Write a recursive function to implement the quicksort algorithm. Write a driver program to accept a list of numbers from the user, sort them, and display the results to standard output. Test your function on integer arrays

of size zero, one, two, and ten, and sixteen. Rubric Grading: - Does the program accept numeric input and display correct results? - Is the function recursive? - Does the function work on inputs of size zero, one, two, ten, and sixteen? - Does the function choose a suitable pivot element? - Is the relative ordering of elements retained as they are copied around the pivot? - Is the program well-documented? - Does the program include a Makefile? - Does the program compile without warnings or errors?

Repeatable

No

Methods of Instruction

- Laboratory
- Lecture/Discussion
- Distance Learning

Lab:

1. Following an instructor discussion on Makefile, the students will construct a Makefile for their own assignments. (Laboratory Objective 14)

Lecture:

1. Building a multi-file project is a complex process. The instructor will lecture and present a project consisting of at least three C files, three headers files, and a library (such as libm). The project can be built "by hand" using four compilation and link commands. From this information, a dependency graph can be illustrated. This dependency graph forms the basis of constructing a Makefile -- a file that describes the sequence of events needed to build a project and automatically re-build only the affected files when changes are made. The instructor will walk the students through writing and testing a Makefile for the example project. (Lecture Objective 3)

Distance Learning

1. In the distance learning format, the instructor prepares a video demonstration using screencasting software of writing a C program that uses the printf and scanf functions to get input from the user, do a calculation, and present the results. Students watch this video to prepare for the week's lab assignment. Example: the student writes a program to calculate the area of a circle given the diameter. (Lecture Objective 1; Laboratory Objective 1)

Typical Out of Class Assignments Reading Assignments

1. Read the textbook chapter about the primitive data types. Pay attention to how each has a different formatting code in the printf function. Be prepared to discuss in class. 2. Read the description of the programming assignment for this week. Prepare to discuss in class how to decompose the problem into functions and what variables will be needed.

Writing, Problem Solving or Performance

1. Write a recursive function to implement the quicksort algorithm. Test your function on integer arrays of size zero, one, two, and ten, and sixteen. 2. Compare and contrast the Java and C programming languages in the following areas: performance, memory usage, portability, ease of readability.

Other (Term projects, research papers, portfolios, etc.)

Numerous hands-on programming assignments.

Required Materials

- C for Programmers with an Introduction to C11
  - Author: Deitel, Paul and Harvey Deitel
  - Publisher: Deitel Developer Series
  - Publication Date: 2013
  - Text Edition: 1st
  - Classic Textbook?: No
  - OER Link:
  - OER:
- Understanding and Using C Pointers
  - Author: Reese, Richard
  - Publisher: O'Reilley
  - Publication Date: 2013
  - Text Edition: 1st
  - Classic Textbook?: No
  - OER Link:
  - OER:

Other materials and-or supplies required of students that contribute to the cost of the course.