C S 2A: OBJECT-ORIENTED PROGRAMMING METHODOLOGIES IN C++

Foothill College Course Outline of Record

Heading	Value
Effective Term:	Summer 2022
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Advisory:	Intermediate Algebra or equivalent; not open to students with credit in C S 2AH.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Area V: Communication & Analytical Thinking
Transferable:	CSU/UC
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to write and debug C++ programs which make use of the fundamental control structures and method-building techniques common to all programming languages. Specifically, the student will use data types, input, output, iterative, conditional, and functional components of the language in his or her programs.
- A successful student will be able to use object-oriented programming techniques to design and implement a clear, well-structured C++ program. Specifically, the student will use and design classes and objects in his or her programs.

Description

Systematic introduction to fundamental concepts of computer science through the study of the C++ programming language. Coding topics include C++ control structures, objects, global-scope functions, class methods, arrays and elementary data structures. Concept topics include algorithms, recursion, data abstraction, problem solving strategies, code style, documentation, debugging techniques and testing.

Course Objectives

The student will be able to:
A. Describe the basic components of the C++ software development environment.
B. Describe the C++ software development life cycle from concept design through documentation, testing and maintenance.
C. Produce clearly written code in an industry standard style appropriate for C++.
D. Define both primitive and compound data types and give examples in C++ of each type.
E. Use C++ variable expressions in a program to compute numeric and string results.
F. Incorporate well-formatted output and user-interaction in a console program.
G. Define, analyze and code the basic C++ conditional and iterative control structures and explain how they can be nested.
H. Design, implement, test, and debug functions and methods that can be used in programs, and demonstrate the way value and reference parameters are passed in such functions and methods.
I. Apply the techniques of structured (functional) decomposition to separate a C++ program into computational and interactive modules.
J. Write C++ programs using object-oriented design, and contrast the difference between object-oriented and procedural code.
K. Discuss the special syntax needed to utilize the "this" pointer and static member methods in C++.
L. Create analytical algorithms that use arrays for solving simple problems.
M. Explain how errors can be reported to the calling function.
N. Explain what an algorithm is and give examples of how algorithms are implemented in a C++ program.
O. Design, implement, test, and debug a C++ program that uses each of the following fundamental programming constructs: string processing, numeric computation, simple I/O, arrays and the C++ API.
P. Solve problems that have origins in a variety of disciplines including math, science, the internet and business.
Q. Explain the difference between syntax and semantics in the context of C++, and place C++ in its historical context among high-level languages.

Course Content

A. The Software Development Environment
1. The C++ run-time environment
2. Integrated development environments (IDEs)
3. Compiled vs. interpreted languages
4. Survey of major languages
5. Obtaining and installing a C++ IDE
B. The Software Development Life-Cycle
1. Overview of design
2. Overview of development
3. Overview of documentation
4. Overview of testing
5. Overview of maintenance
6. Compiler errors vs. run-time errors
7. Debugging strategies
C. Coding Standards, Conventions and Styles
1. Acceptable indentation options
2. Naming conventions for variables and methods
D. Data Types
1. Primitive vs. compound types
2. Numeric types
a. Range and precision
3. Char types
4. C++ strings vs. old-style C char arrays
5. Logical types
6. Array types
7. Type compatibility
8. the const modifier
9. C++-defined classes as compound types
E. Variable Expressions
1. Numeric operators and expressions
2. String operators and expressions
3. Logical operators and expressions
4. Operator precedence
F. Basic Input-Output Strategies
1. Console I/O
2. Use of cin and cout

3. Use of getline()
4. Formatting values for clean output
G. Control Structures
1. Selective and conditional statements
2. Loop statements
3. Nesting levels in control statements
H. Methods and Functional Programming
1. Parameter passing
2. Pass-by-value vs. Pass-by-reference
3. Functional returns
4. Variable scope, binding, lifetime and visibility
5. Default parameters
6. Function overloading
7. Recursion
I. Structured Programming Elements
1. Separation of computation and I/O
2. Modularity
J. Object-Oriented Programming using Classes and Methods
1. Encapsulation of member data
2. Encapsulation of member methods
3. Constructors and destructors
4. Instance vs. static members and methods
5. Initialization of static members
6. Data abstraction as realized through correct selection of member data and methods
7. Data privacy as supported by mutator and accessor methods
8. Object composition (the "has a" relationship)
9. Correct interpretation of assignment between C++ objects
10. Procedural languages vs. object-oriented languages
K. Special C++ Considerations: "this" and static usage
1. Static member initialization syntax
2. Client access of static methods through the scoping operator
3. Use of the "this" pointer and how pointers differ from standard (i.e., non-pointer) variables
L. Arrays
1. Arrays inside classes
2. Arrays of objects
M. Error Reporting
1. Functional returns
2. Reference parameters
3. Avoiding end-use output to report errors in functions or methods
N. Algorithms
1. Role of algorithms in the problem-solving process
2. Simple uses of recursion for divide-and-conquer strategies
3. Simple Sorting via a representative sort algorithm
4. Linear searches
O. Essential Examples and Assignment Areas
1. String/text processing
2. Numeric computation
3. User interaction
4. Arrays
5. Using C++-defined API methods
6. Creating and using a programmer-defined class
P. Applications used Throughout Course in Selected Areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet
Q. History and Syntax
1. C++ compared with prior languages
2. Current language alternatives to C++
3. Examples of how different languages use differing syntax to implement a single semantic construct

Lab Content

A. Familiarization with the beginning-level online lab environment
1. Modify and customize the settings of an Integrated Development Environment (IDE)
2. Use the IDE to create a new programming project
3. Organize projects within an IDE to make submitting labs and switching project environments an orderly process
4. Gain experience with the steps needed to edit a simple program
5. Modify IDE settings to produce an industry standard code style
B. Finding and fixing errors in simple programs
1. Demonstrate the complete edit-compile-run cycle of a simple program using IDE or command-line environment
2. Distinguish between compiler/syntax errors and logic errors
3. Develop strategies for dealing with each type of error
4. Debug code to produce a working program
C. Exploring the different data types using the compiler/IDE
1. Gain experience in effectively using the IDE to create code with primitive numeric types
2. Gain experience in effectively using the IDE to create code with primitive character types
3. Use the IDE to assist in defining and using compound data types
4. Solve syntax and logic problems that arise from typical incorrect formulation of data types
D. Demonstrating user interaction (I/O) through the IDE's console or GUI capabilities
1. Play the role of user and programmer, alternately, to establish a user-interaction plan for a program
2. Evaluate and comment on other students' user-interaction plan
3. Change modes from source code design (editing mode) to end-user interaction (run mode) in your IDE in order to perform Q/A on the program

4. Fix poor interaction behavior by adjusting source code and rerunning program until a satisfactory result is achieved
E. Designing, implementing and testing a program that demonstrates "intelligence" though a combination of control statements
1. Become familiar with selection, loop and nesting to imbue a program with correct logic behavior
2. Use structured programming to make control structures maintainable
3. Run the program multiple times to verify that its control statements produce the correct behavior or output under any scenario
4. Fix incorrect logic behavior by adjusting control structures and rerunning program until a satisfactory result is achieved
F. Incorporating functions and class methods in programming projects
1. Gain experience in designing, implementing and testing a function/method that demonstrates how binding, visibility and variable lifetime work in an OOP language
2. Use a previously written function or method in a client program
3. Refine methods/functions by adding or changing their definitions and locally bound variables, and observe the result
4. Deduce the impact of a function's or method's design on the programs that invoke it
G. Building a program around object-oriented techniques
1. Use previously written classes to instantiate objects in program
2. Use the IDE to assist in the creation of a programmer-defined class
3. Demonstrate the correct choice of class members and methods for each class used

4. Use the IDE's outline view to navigate from one class to another within a program
H. Exploring arrays
1. Understand the proper use of an array
2. Incorporate an array into a program to facilitate the solution of an assigned problem
3. Investigate use of variable indices and loops to shorten and clarify the logic in programs
4. Use debugging techniques to solve problems that arise during the testing of a program
I. Designing, implementing and testing and algorithms
1. Write a program that uses a combination of techniques such as looping, arrays, logic and user I/O, all encapsulated in a coherent algorithm
2. Test the algorithm by running the program multiple times giving it different initial values or inputs
3. Implement a sorting or simple searching algorithm using arrays
4. Transcribe an abstract algorithm into a concrete program that is written and tested using the IDE and submitted online for evaluation

Special Facilities and/or Equipment

A. Access to a computer laboratory with C++ compilers.
B. Website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
C. When taught via Foothill Global Access on the Internet, the college will provide a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the Internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

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

Tests and quizzes
Written laboratory assignments which include source code, sample runs and documentation
Final examination

Method(s) of Instruction

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

Lectures which include motivation for syntax and use of the C++ language and OOP concepts, example programs, and analysis of these programs
Online labs (for all sections, including those meeting face-to-face/on campus), consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet environment. Here, the students will review the specification of each programming assignment and submit their completed lab work
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members
Detailed review of programming assignments which includes model solutions and specific comments on the student submissions
In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs
When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices

Representative Text(s) and Other Materials

Savitch, Walter. Absolute C++, 10th ed.. 2017.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading
1. Textbook assigned reading averaging 30 pages per week.
2. Reading the supplied handouts and modules averaging 10 pages per week.
3. Reading online resources as directed by instructor though links pertinent to programming.
4. Reading library and reference material directed by instructor through course handouts.
B. Writing
1. Writing technical prose documentation that supports and describes the programs that are submitted for grades.

Discipline(s)

Computer Science