C S 2B: INTERMEDIATE SOFTWARE DESIGN IN C++

Foothill College Course Outline of Record

Heading	Value
Effective Term:	Summer 2025
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Prerequisite:	C S 2A.
Advisory:	Demonstrated proficiency in English by placement via multiple measures OR through an equivalent placement process OR completion of ESLL 125 & ESLL 249.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Area 1B: Oral Communication & Critical 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 use the C++ environment to define the basic abstract data types (stacks, queues, lists) and iterators of those types to effectively manipulate the data in his or her program.
- A successful student will be able to write and debug C++ programs which make use of inheritance, i.e., the "is a" relationship, common to all OOP languages. Specifically, the student will define base and derived classes and use common techniques such as method chaining in his or her programs.
- A successful student will be able to define and use C++ templates to make their data and algorithms work with a variety of data types.

Description

Systematic treatment of intermediate concepts in computer science through the study of C++ object-oriented programming (OOP). Coding topics include C++ derived classes, class templates, function templates, virtual functions, operator overloading, an introduction to the Standard Template Library, multiple inheritance, pointers, dynamic memory allocation and file I/O. Concept topics include OOP project design, inheritance, polymorphism, method chaining, functional programming, linked-lists, FIFOs, LIFOs, events in GUIs and guarded code.

Course Objectives

The student will be able to:
A. Configure an Integrated Development Environment (IDE) appropriate for advanced C++ programming.
B. Use both instance members and static members, as appropriate, in class design.
C. Analyze and demonstrate the use of dynamic and static C++ multi-dimensional arrays.
D. Design, implement, and test C++ programs that use class inheritance, and explain why this is an example of the "is-a" relationship.
E. Demonstrate the use of function chaining between derived class and base class methods, and give examples of the C++ syntax used for chained constructors.
F. Apply unary and binary operator overloading to different situations and explain how it simplifies syntax.
G. Describe declaration models for run-time storage allocation, garbage collection, deep memory copies and type checking.
H. Explain how guarded code is implemented in C++ through exceptions.
I. Express numbers in decimal, binary and hexadecimal representations and use bitwise logical operators to process data at the bit and byte level.

J. Demonstrate a working knowledge of basic abstract data types (ADTs) and produce examples of each.
K. Explain what abstract classes and pure virtual functions are and how they are used.
L. Name the "Big Three" class methods in C++ and discuss the role of each.
M. Write code that makes effective use of the Standard Template Library.

N. Define various types of C++ template classes and show how each is specialized to a class by the client program.
O. Explain the use of multiple inheritance.
P. Demonstrate how files are written-to and read-from in C++.
Q. Design, implement, test, and debug intermediate-level C++ programs that use each of the following fundamental programming constructs: string processing, numeric computation, simple I/O, arrays and the C++ API.
R. Write applications that solve problems in one or more application area: mathematics, physics, chemistry, cellular automata, 3-D simulation, astronomy, biology, business, internet.

Course Content

A. Setting up a complete C++ environment
1. The Standard Template Library (C++ collections)
2. Eclipse
3. Configuring the IDE for advanced C++ programming
B. The proper use of class members and methods
1. When to use instance members and methods
2. When to use static members and methods
3. Implicit and explicit use of the "this" pointer
4. The context-sensitive meanings of "const"
C. Multi-dimensional arrays
1. Dynamic allocation pointer-based arrays
2. 2-D arrays
3. Arrays of pointers
D. Inheritance
1. The "is a" relationship
2. Base classes
3. Derived classes (subclasses) and class hierarchy
4. Virtual functions
5. Derived class constructors and destructors
6. Member method overriding vs. simple overloading
7. Private, protected and public members
8. Encapsulation and polymorphism
E. Function chaining
1. Chaining in constructors using initializers
2. Member method chaining
F. Operator overloading
1. Unary operators

2. Binary operators
3. Member vs. global-scope operators
G. Storage allocation methods and deep vs. shallow memory copies
1. Run time binding and storage management of activation records
2. Declaration consequences of pointers, references and value parameters
3. Strong type-checking and run-time vs. compile time error detection
4. Effect of declaration strategy on binding, visibility and lifetime of variables
5. Effect of declaration strategy on scope and persistence of variables
6. Instantiation of member objects in constructors
7. Copy constructors and deep copies of objects
H. Exception handling and event-driven programming
1. Built-in C++ exception classes
2. User-defined exceptions
3. When to re-throw and when to handle an exception
4. Alternatives to exceptions
5. GUI API (instructor's choice) and event-handling techniques
I. Non-decimal arithmetic
1. Bitwise numeric operators
2. Bitwise logical operators
3. Binary and hexadecimal constants
J. Topics in Abstract Data Types (ADTs)
1. The vector ADT and C++'s Vector
2. The linked-list ADT and C++'s List
3. The Stack and Queue ADT
4. Implementing ADTs through inheritance
5. Using existing ADTs from STL
K. Abstract classes
1. Defining and using abstract classes
2. Pure virtual functions
L. Deep copies and the "Big Three"
1. The copy constructor
2. The assignment operator
3. The destructor and cleaning up
M. C++ Standard Template Library
1. C++ Vectors
2. C++ Lists and Iterators
N. C++ Templates
1. Class templates
2. Function templates
3. Type parameters
O. Multiple inheritance
1. Common grandparent classes
2. Distinct grandparent classes
3. Other combinations
P. Topics in C++ file I/O
1. Streams
2. Input streams
3. Output streams
Q. Essential examples and assignment areas
1. String/text processing
2. Numeric computation
3. User interaction
4. Multi-class projects and compound data types
5. Inheritance-based projects
6. Representative GUI project with event-driven design
R. Applications used throughout course in selected areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet

Lab Content

A. Familiarization with the intermediate-level online lab environment
1. Modify and customize project-specific and global settings of an Integrated Development Environment (IDE)
2. Use the IDE to create multi-file programming projects
3. Organize projects within an IDE so as to support easy project-switching and orderly submission of labs
4. Gain experience with the steps needed to edit a complex program
5. Modify IDE settings to produce an industry standard code style
B. Organizing and debugging multi-class projects
1. Demonstrate the ability to debug programs that contain multiple classes
2. Distinguish between interface and implementation in projects by creating classes that are independent of I/O modality
3. Write individual component classes that are independent of client use and can serve several client programs
4. Incorporate symbolic constants, statics and instance members into classes in a way that demonstrates a mature understanding of object-oriented programming (OOP) in a lab project
5. Debug a multi-class project to produce a working program
C. Exploring advanced array constructs in class design
1. Gain experience in effectively using single and multi-dimensional arrays as class members
2. Apply nested loops to process multi-dimensional arrays
3. Use the IDE to debug errors in multi-dimensional arrays
4. Solve problems using fixed-size and dynamic sized arrays, as appropriate
D. Demonstrating competence in intermediate level algorithm design using classes within the IDE
1. Use the IDE to implement a multi-faceted algorithm and/or simulation that makes effective use of OOP
2. Evaluate and comment on other students' algorithms
3. Utilize a combination of string processing and numeric processing to address various aspects of the algorithm implementation
4. Produce clear program runs which demonstrate that the algorithm addresses a variety of cases and/or input states
5. Incorporate bitwise and logical operations to address binary logic tasks within an algorithm
E. Building a program that uses class inheritance to demonstrate how re-use is handled in OOP
1. Create a project that contains at least one class intended to be used as a base class
2. Derive (sub-class) one or more classes from the base class
3. Use function chaining to avoid code duplication between base classes and derived classes
4. Differentiate between, and document in your lab, the distinct use of method overloading and method overriding
F. Incorporating basic abstract data types in programming projects
1. Implement a fundamental abstract data type (ADT) such as a queue or stack in a programming lab
2. Use a previously written ADT from the programming language's application programmer interface (API)
3. Incorporate inheritance in a project that uses ADTs
4. Provide a client program that tests and demonstrates the correct behavior of the ADT
G. The proper use of deep and shallow copies in conjunction with inheritance and other advanced techniques learned in previous labs

1. Create the proper set of methods within a class that enables an object to be cloned (copied deeply) correctly
2. Utilize inheritance to reinforce the segregation of data into base- and derived-level behavior
3. Utilize at least one other lab concept previously, such as binary logic or multi-dimensional arrays to further improve integration of intermediate concepts
4. Separate I/O and implementation in advanced programs
H. Building projects that use generics (AKA templates)
1. Demonstrate the difference in a lab project between deriving from a base class and specializing a generic
2. Practice writing a generic as well as using a language-defined generic (template) in a project
3. Use generics to exercise some aspect of ADTs such as specializing a generic ADT to make its behavior specific to an assigned project specification
4. Employ debugging techniques to solve problems that arise when designing with generic (template) classes
I. Exceptions and file I/O in programming
1. Write a class that has methods which use exception handling to report errors to the client
2. Write a test client that uses try/catch blocks to detect exceptions
3. Implement an algorithm or simulation that reads from and/or writes to files rather than the user screen
4. Demonstrate various ways errors are handled and reported besides exceptions

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.

Weiss, Mark Allen. Data Structures and Algorithm Analysis in C++, 4th ed.. 2013.

The Weiss text is a classic text in the field and is used by many universities in both undergraduate and graduate classes on the subject of data structures.

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.
2. Writing specifications using prose to connect natural English language to the formulaic programming languages.

Discipline(s)

Computer Science