

# CS A250: C++ PROGRAMMING LANGUAGE 2

- 12. Produce programs that show an introductory understanding of functional programming.

Item	Value
Curriculum Committee Approval Date	03/12/2025
Top Code	070600 - Computer Science (Transfer)
Units	4 Total Units
Hours	108 Total Hours (Lecture Hours 54; Lab Hours 54)
Total Outside of Class Hours	0
Course Credit Status	Credit: Degree Applicable (D)
Material Fee	Yes
Basic Skills	Not Basic Skills (N)
Repeatable	No
Open Entry/Open Exit	No
Grading Policy	Standard Letter (S), • Pass/No Pass (B)

## Course Description

Second course in ANSI/ISO Standard C++ programming language. Topics include sorting and searching, data structures, operator overloading, memory management, exception handling, name scope management, polymorphism, templates, STL containers, STL algorithm and iterators, and functional programming. PREREQUISITE: CS A150; and CS A100, CS A122, CS A131 or CS A170. Transfer Credit: CSU; UC.

## Course Level Student Learning Outcome(s)

1. Use C++ to solve problems that incorporate the use of class and function templates, linked lists, and overloaded operators.
2. Use a functional programming language to solve problems by implementing lambda expressions.

## Course Objectives

- 1. Compare performance of various sort and search techniques.
- 2. Create class templates and function templates that show the extensibility of the language.
- 3. Create programs that use the container classes in the Standard Template Library (STL).
- 4. Apply memory management techniques to solve problems.
- 5. Differentiate between the assignment operator and the copy constructor.
- 6. Produce programs that use the concept of overloaded functions and operators with respect to classes.
- 7. Implement and manipulate singly- and doubly-linked lists.
- 8. Produce programs that show the understanding of polymorphism as it applies to C++.
- 9. Apply the concept of abstract classes to enhance their understanding of OOP concepts.
- 10. Produce programs that use the exception handling features of the language.
- 11. Produce programs that show an understanding and appreciation for iterators and algorithms in STL.

## Lecture Content

Separate compilation Principles of Encapsulation Header files Implementation files Reusable components Using #ifndef Algorithm efficiency Big-O notation Estimate the efficiency of algorithms Compare the performance of algorithms Compare sequential search and binary search Linked data representations Standard lists and iterators Implement singly- and doubly-linked lists Insert and remove nodes in singly- and doubly-linked lists Stacks and queues STL common containers and algorithms Operator overloading Overloading simple arithmetic operators Overloading comparison operators Overloading input and output Overloading increment and decrement operators Overloading the assignment operator Memory management Different categories of memory Constructors, destructors and copy constructors Reference counting Exception handling Various methods to handle exceptions Throw, catch and try block Throwing objects Stack unwinding Name scope management Name scope and lifetime Forward reference Protected scope Friends Nested classes Private inheritance Name spaces Polymorphism Polymorphic variables Virtual and non-virtual overriding Pure virtual member functions Run-time typing information Multiple inheritance Templates Template functions Template classes Typedef Non-type template arguments Non-type template arguments Standard template library containers Fundamental sequential containers vector, list, deque Adapted containers stack, queue, priority queue Ordered container set Associative container map Standard template library iterators and algorithms Iterator as a general idea for pointers Categories of iterators Function objects, generators, predicates Generic algorithm Stream iterators Functional programming Installing scheme Prefix notation Defining variables and procedures Cond and if Logical operators Constructing procedures using Lambda

## Lab Content

The following programming labs are designed to cover various aspects of the course content, providing hands-on experience and practical understanding of C++ programming concepts, from memory management to advanced topics like templates, polymorphism, and functional programming: Introduction to Separate Compilation and Encapsulation Principles: Implement a basic C++ program using separate compilation techniques. Develop header and implementation files for a reusable component, emphasizing encapsulation principles. Algorithm Efficiency and Linked Data Representations: Estimate the efficiency of an algorithm using Big-O notation. Implement singly- and doubly-linked lists, showcasing insertion and removal of dynamic nodes. Memory Management Techniques: Explore memory management techniques by implementing constructors, destructors, copy constructors, and copy assignment operators for classes that create objects pointing to dynamic data. Operator Overloading and Exception Handling: Overload simple arithmetic operators, prefix and postfix operators, insertion and extraction operators, and comparison operators for a custom class. Polymorphism and Template Concepts: Implement polymorphic variables using both virtual and non-virtual overriding. Create template functions and classes. STL and Iterators: Use and modify fundamental sequential containers (vectors, lists), adapted containers (stacks, queues, priority queues) and ordered containers (sets, multisets, maps, multimaps) from the STL. Understand and implement different categories of iterators. Generic Algorithms: Work with function objects, generators, and predicates in generic STL algorithms. Advanced Concepts: Explore multiple inheritance and private inheritance. Functional Programming: Write functional

programs using Racket or an equivalent functional programming language.

## Method(s) of Instruction

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

## Instructional Techniques

Lecture/discussion

## 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

Written examination and laboratory exercises.

## Required Writing, Problem Solving, Skills Demonstration

Successful performance of the laboratory assignments.

## Eligible Disciplines

Computer science: Master's degree in computer science or computer engineering OR bachelor's degree in either of the above AND master's degree in mathematics, cybernetics, business administration, accounting or engineering OR bachelor's degree in engineering AND master's degree in cybernetics, engineering mathematics, or business administration OR bachelor's degree in mathematics AND master's degree in cybernetics, engineering mathematics, or business administration OR bachelor's degree in any of the above AND a master's 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. Master's degree required.

## Textbooks Resources

1. Required Deitel, P., Deitel, H.. C++ How to Program: An Objects-Natural Approach, ed. Pearson, 2023