Learning Objectives

Identify and overcome the difficulties encountered by students when learning how to program
List and explain the software development roles played by students
List and explain the phases of the tight spiral software development methodology
Employ the concept of the Flow to tap creative energy
List and explain the primary areas of the Project Approach Strategy
State the purpose of a header file
State the purpose of an implementation file
Explain the importance of separating interface from implementation
Employ multi-file programming techniques to tame project complexity
Explain the use of #ifndef, #define, and #endif preprocessor directives
Apply preprocessor directives to implement multi-file programming projects
State the importance of adopting consistent variable and constant naming conventions
List and describe the two types of C++ comments
List and describe the steps of the program creation process to include creating source code files, preprocessing, compiling, and linking
List the input and output to each stage of the program creation process
List and describe the primary functions of an Integrated Development Environment (IDE)
Describe the concept of a project
List and describe the steps required to create projects using Macintosh, Windows, and Unix development environments
Demonstrate your ability to create projects in the IDE of your choice
State the purpose of the UNIX make utility
State the purpose of a UNIX makefile
Demonstrate your ability to create and use UNIX makefiles
Utilize Metrowerks CodeWarrior to create projects on Macintosh™ and PC platforms
Utilize Tenon Intersystems' CodeBuilder™ to create projects on the Macintosh™ platform
List and describe the similarities between different UNIX development environments
Apply the project approach strategy to help you systematically implement a program that satisfies the requirements of a given project specification
Iteratively apply the development cycle to help you implement your programming projects
List and describe the phases of the Project Approach Strategy
List and describe the steps of the software development cycle
List and describe the different development roles performed during the development cycle
Translate a project specification into a software design that can be implemented in C++
Implement a software design in C++ using a functional decomposition approach
List and describe the steps involved with functional decomposition
Describe how the development cycle can be employed in a tight spiral fashion
State the importance of compiling and testing early during the development process
Define the concept of a computer
Explain why the computer is a remarkable device
Explain how a computer differs from other machines
Explain how a computer stores and retrieves programs for execution
State the difference between a computer and a computer system
State the purpose of a microprocessor and the role it plays in a computer system
Define the concept of a program from the human perspective and the computer perspective
Describe how programs are represented in a computer's memory

List and describe the nine stages of the C++ program transformation process
List and describe the four steps of the processing cycle
State the purpose and objective of a computer's memory system
Define the concept of an algorithm
List the characteristics of a good algorithm
Describe what constitutes a minimum well-formed C++ program
List the keywords reserved for use by the C++ language
State the purpose of variables, constants, expressions, and statements
Demonstrate your ability to declare, define, and use variables
Demonstrate your ability to declare, define, and use constants
List and describe the purpose of the C++ fundamental data types
Determine data type sizes with the sizeof operator
Utilize variables and constants in simple C++ programs
List the native C++ operators and state their precedence
Write C++ programs using simple and compound statements
Describe variable scoping and state how the block structure of C++ can affect variable visibility
Utilize simple input and output techniques using the cin and cout objects
Describe the required parts of a minimal C++ program
Utilize an IDE's disassembly tool to gain deeper understanding of C++ program structure
List and describe the parts of a typical C++ program to include source files, main() function, library files, and preprocessor directives
Control the flow of program execution with C++ control flow statements
State the purpose and use of the if statement
Explain the purpose of a null statement
Utilize blocks to create local variable scopes in control statements
State the purpose and use of nested if statements
State the purpose and use of the for statement
State the purpose and use of nested for statements
State the purpose and use of the keywords break and continue
State the purpose and use of the while statement
State the purpose and use of the do statement
State the purpose and use of the switch statement
Explain the importance of using break to exit case statements properly
Explain the importance of a default case
Demonstrate your ability write effective, self-commenting expressions utilizing sound identifier naming techniques
State the purpose and use of pointers and references in C++
State the definition of an object
Explain how to determine an object's address using the & operator
Explain how to declare pointers using the pointer declarator *
Explain how to dereference a pointer using the * operator
Describe the concept of dynamic memory allocation
Explain how to dynamically create objects using the new operator
Explain how to destroy objects using the delete operator
Explain how to declare references using the reference declarator &
Explain why references must be defined at the point of declaration
Describe the benefits of using references vs. pointers
Utilize pointers and references to create powerful C++ programs
Describe the concept of an array
State the purpose and use of single- and multi-dimensional arrays
Describe how to declare and initialize single- and multi-dimensional arrays
Explain how the compiler uses the array's declared type to calculate offset addresses into an array
Explain how to access array elements using array subscript notation and pointer notation
List and describe the similarities between an array name and a pointer
Explain how to use pointers to dynamically allocate memory for an array with the new[] operator

Explain how to release dynamically allocated array memory with the delete[] operator
Explain how to idiomatically process an array using a for loop
Explain how to process multi-dimensional arrays using nested for loops
Explain how strings are implemented in C++
Utilize single- and multi-dimensional arrays in your C++ programming projects
State the purpose and use of functions in C++
Explain how to declare and define functions
State the purpose and use of function return types
State the purpose and use of function parameters
Describe the concept of function calling
Explain the use of local function variables and their scoping rules
Describe how to pass arguments to a function by value and by reference
Describe how to maximize function cohesion and minimize coupling
Describe the concept of function signatures
Describe how to overload functions
Explain the concept of recursion
Explain the concept and use of function pointers
Explain how to create function libraries
Utilize functions in your C++ programming projects
Describe how functions are used to modularize C++ program functionality
Demonstrate your ability to minimize function coupling and maximize function cohesion in C++ programming projects
Create new data types to improve problem abstraction
Use the typedef keyword to create type synonyms for existing data types better suited to the problem domain
Explain how type synonyms can be used to improve program maintainability and readability
Create and use enumerated data types in your programming projects
Describe the default enum state values and explain how they can be changed
Explain how to resolve enum state name conflicts
Create and use structures in your programming projects
Explain how to use the dot operator to access structure and class elements
Create and use simple classes in your programming projects
State the difference between structures and classes
Describe when you would want to use structures vs. classes in a programming project
List the key differences between structures and classes
State the purpose and use of the this pointer
List and define the following terms: class, base class, derived class, superclass, subclass, abstract base class, virtual function, object, message passing, OOA&D, inheritance, data encapsulation, interface, & implementation
State the purpose and use of the class construct in C++
List and describe the parts of a class declaration
State the importance of the terminating semicolon of a class declaration
Explain how to use access specifiers to control horizontal member access
State the function and purpose of constructors
State the purpose and use of overloaded constructors
Explain how to overload constructors
Explain how to use the initializer list to initialize class attributes
State the purpose and use of destructors
Explain how to overload class member functions
Explain the importance of separating the class interface from its implementation
Explain how to call class member functions from within class member functions
Utilize complex class constructs in your C++ programming projects
Utilize initializer lists to initialize class attributes
List and define the following terms: constructor, destructor, default constructor, overloaded constructor, and overloaded functions
Explain how to design complex classes using user-defined abstract data types
Describe the concept of aggregation

State the relationship between aggregation and object lifetime
Explain the difference between contains by value and contains by reference
Describe the concept of simple aggregation
Describe the concept of composite aggregation,
Explain how to implement message passing between objects
Explain how to utilize pointers and references in the design of complex classes
Explain how to express aggregation in UML notation
State the purpose and use of a UML sequence diagram
Demonstrate your ability to use simple and composite aggregation to implement C++ programming projects
State the purpose and use of inheritance in C++ class design
Explain how to apply the three access specifiers, public, protected, and private
Explain how to hide base class functions with derived class functions
Explain how to call a base class constructor from a derived class initializer list
Explain the use of the virtual key word as it relates to destructors and class member functions
Explain how to override virtual base class functions
Explain how to implement pure virtual functions
Explain how to declare and use abstract base classes
Explain how to substitute derived class objects where base class objects are specified
Explain how to implement multiple inheritance
Explain what is meant by virtual base class
Explain how to safely use inheritance in your application design
Explain how to extend the UML class diagram to illustrate class inheritance hierarchies
Demonstrate your ability to express inheritance with a UML class diagram
Demonstrate your ability to utilize inheritance in the design of complex C++ programming projects
Define the term Ad Hoc Polymorphism
Explain how to achieve ad hoc polymorphic behavior through operator overloading
Identify which C++ operators can be overloaded
Demonstrate your ability to overload the following arithmetic operators: +, -, *, /
Demonstrate your ability to overload the following relational operators: <, >, <=, >=
Demonstrate you ability to overload the following equality operators: ==, !=
Demonstrate your ability to overload the following unary operators: prefix ++, postfix ++, prefix -, postfix -
Demonstrate your ability to overload the subscript operator: []
Demonstrate your ability to overload iostream operators
Explain when and how to use friend functions to implement operator overloading
Explain when overloaded operator functions should be class members
Explain why and when operator overloading is right for your design
Explain how to achieve static polymorphic behavior through the use of templates
Explain how to write generic code using templates
Describe the concept of a template class
Explain how to declare and implement function templates
Explain how to declare and implement class templates
Explain how to declare and implement class member function templates
Demonstrate your ability to declare and implement single parameter template classes
Demonstrate your ability to declare and implement multiple parameter template classes
Explain how to use components of the C++ Standard Template Library in your C++ programming projects
State the purpose and use of STL iterators, algorithms, and containers
Demonstrate your ability to utilize class and function templates to create generic code in support of your C++ programming projects
State the definition of dynamic polymorphism
Explain how to achieve dynamic polymorphic behavior through the use of base class pointers and derived class objects
State the importance of abstract base classes in object-oriented design
Describe the role virtual functions play in implementing dynamic polymorphic behavior
State the purpose and use of virtual destructors
Describe the concept of pure virtual functions

State the purpose and use of abstract base classes
State the importance of a consistent derived class interface and the role it plays in achieving robust polymorphic behavior
Explain why polymorphic behavior is a critical component of good object-oriented design
Demonstrate your ability to utilize dynamic polymorphism in your C++ programming projects
List and define the following terms: base class, abstract base class, virtual function, pure virtual function, virtual destructor, inheritance hierarchy, base class pointer, derived class object, and dynamic polymorphic behavior
State the important role well-behaved objects play in good object-oriented design
List and describe the functions required to get user defined objects to behave like native types
List and describe the four minimum functions required to implement the orthodox canonical class form
Demonstrate your ability to utilize the orthodox canonical class form in your C++ programming projects
Demonstrate your ability to extend the orthodox canonical class form to suit the needs of a particular class
Explain why compiler-supplied constructors and destructors may not provide appropriate object behavior for complex class types
List and define the following terms: orthodox canonical class form, default constructor, destructor, copy assignment operator, copy constructor
Explain how to create and integrate assembly language object modules
Explain how to integrate legacy C code
State the purpose and use of the extern keyword
Explain why the extern keyword is necessary to link to legacy C code modules
Describe the concept of name mangling
Explain how to call C and C++ routines from Java applications
List the steps required to create, compile, and link to an assembly language module
List the steps required to create a Java JNI project and call a C++ native method from a Java program
State the purpose and use of the javah command line tool
Demonstrate your ability to utilize assembly language routines in your C++ programming projects
Demonstrate your ability to call native C++ functions from Java programs
Demonstrate your ability to use inline assembly in a macintosh environment
Demonstrate your ability to use inline assembly in a PC environment
List the preferred characteristics of an object-oriented application architecture
State the definition of the Liskov Substitution Principle (LSP)
State the definition of Bertrand Meyer's Design by Contract (DbC)
Recognize the close relationship between the Liskov Substitution Principle and Design by Contract
Specify preconditions and postconditions for class and instance functions
Specify class invariants
State the definition of the Open-Closed Principle (OCP)
State the definition of the Dependency Inversion Principle (DIP)
Apply the Liskov Substitution Principle in the design and implementation of a class inheritance hierarchy
Apply Design by Contract in the design and implementation of a class inheritance hierarchy
Apply the Open-Closed Principle in the design and implementation of a class inheritance hierarchy
Apply the Dependency Inversion Principle in the design and implementation of a class inheritance hierarchy
State the purpose of a UML modeling tool
List key UML modeling features supported by Embarcadero Technologies' Describe™
Utilize use-case, sequence, and class diagrams to analyze and design a solution to a given programming problem
Utilize Describe™ to develop a solution to a given programming problem up to the point of C++ code generation
Select the appropriate UML diagram based on the corresponding problem analysis or design phase
Employ the UML constructs of aggregation and generalization to create complex class relationships
Utilize Describe™ to reverse engineer existing C++ source code
Utilize Describe™ to generate a web-based project report