College of San Mateo
Official Course Outline

1. **COURSE ID:** CIS 278 **TITLE:** (CS1) Programming Methods: C++ **C-ID:** COMP 122
   Units: 4.0 units **Hours/Semester:** 48.0-54.0 Lecture hours; 48.0-54.0 Lab hours; and 96.0-108.0 Homework hours
   Method of Grading: Grade Option (Letter Grade or Pass/No Pass)
   Prerequisite: MATH 120, CIS 254
   Recommended Preparation:
   Eligibility for ENGL 838 or ENGL 848 or ESL 400.

2. **COURSE DESIGNATION:**
   Degree Credit
   Transfer credit: CSU; UC
   AA/AS Degree Requirements:
   CSM - GENERAL EDUCATION REQUIREMENTS: E2c.Communication and Analytical Thinking
   CSU GE:
   CSU GE Area B: SCIENTIFIC INQUIRY AND QUANTITATIVE REASONING: B4 - Mathematics/Quantitative Reasoning

3. **COURSE DESCRIPTIONS:**
   Catalog Description:
   Object-oriented programming methodology for both computer science majors and computer professionals. Systematic approach to design, construction, and management of computer programs; emphasizing program documentation, testing, debugging, maintenance and software reuse. Also includes UML, virtual machines, exception handling, sorting and searching algorithms, recursion, fundamental graphics, and computer ethics. This course conforms to the ACM CSI standards. A materials fee as shown in the Schedule of Classes is payable upon registration.

4. **STUDENT LEARNING OUTCOME(S) (SLO'S):**
   Upon successful completion of this course, a student will meet the following outcomes:
   1. Demonstrate knowledge and understanding of the principal object-oriented programming concepts.
   2. Implement a medium-size computer program that is stylistically and functionally correct, based on an object-oriented design model.
   3. Reuse existing components through inheritance and polymorphism.
   4. Implement, test, and debug simple recursive functions.
   5. Demonstrate different forms for binding, visibility, scope and lifetime management.
   6. Employ components in the C++ Standard Template Library (STL).
   7. Utilize exception handling to provide a robust computer application
   8. Relate the development of high level languages to the programming paradigms used today

5. **SPECIFIC INSTRUCTIONAL OBJECTIVES:**
   Upon successful completion of this course, a student will be able to:
   1. Demonstrate knowledge and understanding of the principal object-oriented programming concepts.
   2. Implement a medium-size computer program that is stylistically and functionally correct, based on an object-oriented design model.
   3. Reuse existing components through inheritance and polymorphism.
   4. Implement, test, and debug simple recursive functions.
   5. Demonstrate different forms for binding, visibility, scope and lifetime management.
   6. Employ components in the C++ Standard Template Library (STL).
   7. Utilize exception handling to provide a robust computer application
   8. Relate the development of high level languages to the programming paradigms used today

6. **COURSE CONTENT:**
   Lecture Content:
   INTRODUCTION:

   - History of High Level Languages
   - Programming Paradigms
   - C++ and Object Oriented Programming

- Translation: Compilers vs Interpreters
- Execution: Compilers and Linkers
- Portability

ALGORITHMS AND PROBLEM SOLVING:

- Development of an algorithm
- Coding an algorithm
- Debugging strategies (output stmts, IDE debugger)

C++ FUNDAMENTALS:

- Data Types and Internal Representation
- Variables, Expressions and Assignment Statements
- Variable scope and binding
- Type checking in C++
- Control Structures
- I/O streams cout, cerr, cin ; Libraries and namespaces

FUNCTIONS :

- Predefined functions
- Programmer defined functions
- Call by ref/call by val
- Function prototypes
- Separate compilation
- Scope, lifetime and visibility local variables
- Function overloading
- Default arguments
- Recursive algorithms/functions

C++ Arrays

- Declaration, element access, traversal
- Array parameters
- Partially filled arrays
- Simple searching/sorting
- STL vector class

C++ CLASSES and OBJECTS

- UML class diagrams
- Public and private members
- Overloading constructors
- Const functions
- Operator overloading (member, friend and non-member)
- Derived classes and function redefinition
- Static vs dynamic binding
- Polymorphism – virtual functions and abstract classes

C++ Strings

- C-strings
- String class

Pointers and Dynamic Memory Allocation

- Dynamic arrays
- Pointers to objects
- Classes and Dynamic Memory
- Copy constructor, = overload, Destructor

FILE I/O

- Reading from and creating simple text files

- Object Persistence thru Files

EXCEPTION HANDLING

- Creating exception classes
- When/how to throw/catch an exception

TEMPLATE

- Classes and Methods

SOFTWARE ENGINEERING ISSUES

- Requirements and specifications
- Design for reuse
- Iterative development and testing

Lab Content:

Students complete small programming exercises which relate to the current class lecture. Lab exercises involve:

1. Using Microsoft Express (or current IDE) to create a simple project
2. Writing a program which uses C++ I/O, iostream library and namespace directive
3. Writing a program which uses C++ predefined library functions (such as rand and srand)
4. Coding two file program, one file containing a function requiring call by ref parameters, the other file containing an application which calls that function
5. Writing a recursive function and an app which verifies it's correctness
6. Debugging an existing recursive function
7. Writing a program which uses a static array for storage (involves array storage, retrieval and update)
8. Writing a program which uses a partially filled array to store data
9. Writing a program which requires input and modification of a C-string, requiring use of the library
10. Coding a simple class with constructor, mutators and access functions, and an app that uses an object of that type
11. Modifying a class to include functions with object parameters and static functions
12. Coding erroneous test responses and debugging them
13. Modifying a class to include binary and unary operator overload functions as class members
14. Modifying a class to include binary and unary operator overload functions as friend functions
15. Modifying a class to include binary and unary operator overload functions as non-member, non-friend functions
16. Coding a class which requires a dynamic array member variable, and providing copy constructor, =operator overload and destructor
17. Given a class, coding a class which derives from that class, providing constructors, mutators and access methods
18. Modifying a derived class to override base class functions
19. Modifying a base class to include virtual and pure virtual functions, adjusting derived classes accordingly
20. Given an existing application, add exception handing capabilities
21. Writing a simple program which reads data from a file, and produces a copy of that file
22. Writing a program which reads object input from a file, stores and updates object, and stores data back to file
23. Code a couple template functions
24. Use functions from the STL library in an application
25. Code a simple template class and app

7. **REPRESENTATIVE METHODS OF INSTRUCTION:**

Typical methods of instruction may include:

A. Lecture
B. Lab
C. Activity
D. Discussion
E. Individualized Instruction

F. Observation and Demonstration
G. Other (Specify): Lectures, to introduce new topics; “Models” for problem-solving techniques; Class (group) problem solving , each person contributing a potential "next step"; Student participation in short in-class projects; Q/A sessions with students providing both the questions AND the answers; Students working in small groups to solve significant programming assignments. Live code development/debugging demonstrations.

8. **REPRESENTATIVE ASSIGNMENTS**

Representative assignments in this course may include, but are not limited to the following:

Writing Assignments:

The primary writing opportunity for students in this course is documentation supporting their programming projects. This includes both technical documentation targeting a peer audience, and user documentation targeting those using the software the student develops. The technical documentation describes the problem to be solved, the scope of the project, an overview of the solution, and any limitations of the solution. The user documentation is primarily instructional.

Reading Assignments:

Students complete weekly reading assignments from the course textbook and reference online C++ language website references.

Other Outside Assignments:

Textbook exercises and weekly programming assignments comprise the majority of the out-of-class assignments. At least one of the programming assignments is a small-group project to provide experience in a realistic program development environment. Specifically, the intent is to provide an opportunity for students to improve their communication skills and learn to work in a cooperative environment. Faculty may also use "pair-programming" to provide the "real world" development environment.

In addition to lab assignments (described in lab content section), students complete medium size projects:

Project 1 - application which reviews memory management, control structures, and user interface skills from previous course. Includes C++ iostream I/O and namespaces.

Project 2 - application which utilizes both C++ predefined functions and user defined functions

Project 3 - application which employs a partially filled array to store data for a non-trivial problem; C-string technique also required

Project 4 - create a C++ class including overloaded constructors, default arguments, mutators, accessors, worker functions and output functions. Application to use class also required.

Project 5 - create a C++ class together with member and non-member operator overloads of binary and unary operators. String class usage included. Application to use class also required.

Project 6 - create a C++ class with dynamic array member variable, overload and override functions and derive a simple class from this base class. Application to use class also required.

Project 7 - code a C++ abstract class with at least 2 derived classes, and an application which requires storing objects of base type in an array

Project 8 - code a C++ template class and utilize instances of this class in an application which requires exception handling and file I/O for data persistence.

9. **REPRESENTATIVE METHODS OF EVALUATION**

Representative methods of evaluation may include:

A. Class Participation
B. Exams/Tests
C. Group Projects
D. Lab Activities
E. Projects
F. Quizzes
G. Assessment of student contributions during class discussion and project time; Individual programming

assignments;
Midterm and Final exams: short answers from textbook material, general problem solving (similar to in-class work), short program segments (similar to programming assignments);
Assessment of group participation on course projects, including peer-assessment of participation and contribution to the group effort.

10. **REPRESENTATIVE TEXT(S):**
Possible textbooks include:
    A. Stroustrup. *A Tour of C++*, 2nd ed. Addison-Wesley Professional, 2018
    B. Deitel & Deitel. *C++: How To Program*, 10th ed. Pearson, 2016
    C. Lospinoso. *C++ Crash Course*, 1st ed. No Starch Press, 2019
    D. Murach. *Murach's C++ Programming*, 1st ed. Murach, 2018

Origination Date: October 2018
Curriculum Committee Approval Date: November 2018
Effective Term: Fall 2019
Course Originator: Melissa Green