Object-Oriented Programming - Fall 2026

1. Course Information

Contact Information

- **Instructor:** Dr. Terry Griffin
- **Office:** Bolin Hall 124-F
- **Office Hours:** MW: 2:30 p.m. – 4:00 p.m.; TTh: 11:00 a.m. – 12:00 p.m.
- **Office Phone:** (940) 397-4439
- **Email:** terry.griffin@msutexas.edu

Important Dates, Times, and Location

- **Course:** CMPS 2143 — Object-Oriented Programming
- **Days:** Monday / Wednesday / Friday
- **Time:** 11:00 a.m. – 11:50 a.m.
- **Location:** Bolin 103
- **Semester Start:** Monday August 23rd
- **Semester End:** Friday December 4th
- **Labor Day:** Monday September 7th
- **Thanksgiving Holiday:** Wednesday November 25th – Sunday November 29th
- **Last Day for "W":** Monday November 23rd
- **Final Exam:** Monday December 7th, 10:30 a.m. – 12:30 p.m.

Course Communication

- **Course Communication Platform:** Discord
- **Zoom:** https://msutexas-edu.zoom.us/j/9403974439
- **GitHub:** Course repository information will be provided in class.

Course announcements, assignment information, questions, and class discussion may be distributed through the course communication platform and GitHub.

Textbook and Course Materials

Required Textbook: None

No textbook is required. Course materials will be provided by the instructor or made available through course resources.

Students will need access to a computer capable of compiling and running modern C++ programs, Git/GitHub, and other development tools introduced during the semester.

Prerequisites and Expected Background

This course assumes prior programming experience.

Students should enter the course with experience in:

- variables, expressions, conditionals, loops, and functions;
- arrays and basic data structures;
- pointers and references;
- linked versus array-based structures;
- basic classes and member functions; and
- introductory C++ programming.

---

2. Course Overview

Course Description

This course develops object-oriented programming and software design skills using C++.

The course emphasizes the design of user-defined types that behave predictably and integrate naturally with the language and standard library. Students will study abstraction, encapsulation, object lifetime, resource management, operator overloading, composition, inheritance, polymorphism, generic programming, and modern C++ design practices.

Object-oriented programming will be presented as one collection of tools for software design rather than as a required solution for every programming problem. Particular attention will be given to choosing appropriate abstractions and relationships among program components.

Course Approach

A central theme of the course is the design of **useful user-defined types**.

C++ allows programmers to create types that can behave much like types built into the language. A well-designed class can be constructed, copied, moved, compared, printed, stored in containers, passed to algorithms, and used naturally through overloaded operators.

The course will therefore emphasize not simply how to create classes, but how to design classes with interfaces that are natural, predictable, and appropriate for the abstraction being represented.

Inheritance and runtime polymorphism are important object-oriented techniques, but they are not appropriate solutions for every problem. Students will compare inheritance with composition, generic programming, and other approaches and will be expected to select techniques based on the needs of the problem.

The objective is not to maximize the number of classes or inheritance relationships in a program.

The objective is to create useful abstractions.

---

3. Learning Objectives

By the end of the course, students should be able to:

1. Design classes that represent meaningful software abstractions.
2. Distinguish between a class's public interface and its implementation.
3. Use constructors, access control, and class invariants to maintain valid object state.
4. Explain object lifetime, copying, assignment, movement, and destruction.
5. Apply RAII and modern C++ ownership techniques.
6. Explain and apply the Rule of Three, Rule of Five, and Rule of Zero.
7. Overload operators appropriately for user-defined types.
8. Design types with natural and predictable interfaces.
9. Distinguish among association, aggregation, composition, and inheritance.
10. Select composition or inheritance based on the relationship being modeled.
11. Implement runtime polymorphism using virtual functions and abstract interfaces.
12. Use templates and generic programming techniques.
13. Integrate user-defined types with standard containers and algorithms.
14. Recognize common problems involving ownership, object slicing, coupling, and poor abstraction.
15. Refactor object-oriented code toward clearer and more maintainable designs.
16. Evaluate whether an object-oriented solution is appropriate for a given problem.
17. Explain and defend significant software-design decisions.

---

4. Course Content

Major Course Themes

User-Defined Types and Abstraction

Classes will be treated as mechanisms for defining new types rather than simply grouping variables and functions.

Topics may include:

- classes and objects;
- `struct` versus `class`;
- abstraction;
- encapsulation;
- public and private interfaces;
- access control;
- member functions;
- static members;
- class invariants; and
- `const` correctness.

Object Construction, Lifetime, and Ownership

Students will study how objects are created, copied, moved, and destroyed.

Topics may include:

- constructors;
- member initializer lists;
- destructors;
- object lifetime;
- pointers and references;
- dynamic memory;
- ownership;
- copy constructors;
- copy assignment;
- move constructors;
- move assignment;
- shallow versus deep copying;
- RAII;
- smart pointers;
- Rule of Three;
- Rule of Five; and
- Rule of Zero.

Operator Overloading and Natural Interfaces

C++ allows user-defined types to participate in many of the same operations as built-in types.

Topics may include:

- function overloading;
- arithmetic operators;
- comparison operators;
- assignment operators;
- stream insertion and extraction;
- indexing;
- conversion constructors;
- conversion operators;
- `explicit`; and
- three-way comparison where appropriate.

Operator overloading will be used to develop types whose operations are consistent with the abstraction they represent.

Object Relationships and Composition

Larger programs are constructed by combining objects with different responsibilities.

Topics may include:

- association;
- aggregation;
- composition;
- dependency;
- delegation;
- ownership relationships; and
- dependency injection.

Composition will be compared with inheritance as an alternative method for constructing larger systems from smaller components.

Inheritance and Runtime Polymorphism

Inheritance will be studied as one mechanism for representing type relationships and supporting runtime polymorphism.

Topics may include:

- base and derived classes;
- protected members;
- overriding;
- virtual functions;
- pure virtual functions;
- abstract classes;
- virtual destructors;
- base-class references and pointers;
- object slicing;
- `override`;
- `final`; and
- substitutability.

Students will distinguish between inheritance used to represent meaningful type relationships and inheritance used only for code reuse.

Generic Programming

C++ templates provide another form of polymorphism that does not depend on inheritance.

Topics may include:

- function templates;
- class templates;
- template parameters;
- generic algorithms;
- compile-time polymorphism;
- type requirements; and
- C++ concepts where appropriate.

Students will compare runtime polymorphism with compile-time generic programming.

Standard Library Integration

The C++ standard library provides examples of reusable types and algorithms designed around common interfaces.

Topics may include:

- `vector`;
- `list`;
- `deque`;
- stacks and queues;
- maps and sets;
- iterators;
- algorithms;
- lambdas;
- ranges where appropriate; and
- using user-defined types with standard containers and algorithms.

Object-Oriented Design

The course will introduce principles intended to improve maintainability and reduce unnecessary complexity.

Topics may include:

- cohesion;
- coupling;
- separation of concerns;
- composition versus inheritance;
- selected SOLID principles;
- refactoring;
- code smells;
- selected design patterns; and
- unit testing.

Design principles and patterns will be treated as tools rather than fixed rules.

Supporting Topics

The following topics may be introduced as needed within assignments and projects:

- exception handling;
- file I/O;
- UML and simple design diagrams;
- testing;
- debugging;
- header and implementation file organization;
- build tools;
- Git/GitHub workflow; and
- development environment configuration.

Tentative Course Progression

The exact sequence and pace may change based on class progress, assignments, projects, student questions, and topics that arise during the semester.

Part I — Classes and User-Defined Types

- Classes and objects
- Encapsulation and abstraction
- Public interfaces
- Constructors
- `const` correctness
- Class invariants

Part II — Value Types and Natural Interfaces

- Function overloading
- Operator overloading
- Assignment
- Comparison
- Stream operators
- Conversions

Part III — Lifetime and Resource Management

- Destructors
- Copy semantics
- Dynamic resources
- Ownership
- RAII
- Smart pointers
- Move semantics
- Rule of Three / Five / Zero

Part IV — Object Relationships

- Association
- Aggregation
- Composition
- Delegation
- Dependencies
- Ownership relationships

Part V — Inheritance and Polymorphism

- Base and derived classes
- Overriding
- Virtual functions
- Abstract classes

- Runtime polymorphism
- Substitutability
- Object slicing
- Composition versus inheritance

Part VI — Generic Programming and the Standard Library

- Templates
- Generic algorithms
- Containers
- Iterators
- Lambdas
- Concepts where appropriate

Part VII — Software Design

- Coupling and cohesion
- Refactoring
- Selected SOLID principles
- Selected design patterns
- Testing
- Program architecture

---

5. Coursework and Assessment

Programming Assignments

Programming assignments will emphasize both correctness and software design.

Students will develop programs that demonstrate concepts such as:

- meaningful abstractions;
- appropriate class interfaces;
- constructors and object lifetime;
- operator overloading;
- resource ownership;
- composition;
- inheritance and polymorphism;
- templates;
- standard library integration; and
- maintainable program organization.

Assignments must compile and run successfully.

Students are responsible for understanding and being able to explain submitted code.

Projects

A programming project will require students to combine several major course concepts into a larger software solution.

Project requirements may include:

- multiple interacting classes;
- appropriate object relationships;
- operator overloading;
- composition or inheritance;
- polymorphism;
- templates or standard library components;
- testing;
- documentation; and
- GitHub-based submission.

Project-specific requirements will be provided separately.

Quizzes

Quizzes are normally given on Fridays.

Quizzes may cover:

- recent lecture material;
- programming assignments;
- assigned readings or examples;
- C++ syntax and behavior; and
- conceptual design questions.

Quiz scheduling may change when necessary because of examinations, holidays, projects, or changes in course pace.

Examinations

Three major examinations are planned during the semester.

They are tentatively positioned near:

- **Exam 1:** approximately Week 4
- **Exam 2:** approximately Week 8
- **Exam 3:** approximately Week 12

Examinations may include:

- conceptual questions;
- code reading;
- code tracing;
- short programming problems;
- class-design questions;
- comparisons among design alternatives; and
- material drawn from programming assignments.

Study guides will identify the major material students should prepare for each examination.

Final Examination

The final examination will be comprehensive and will emphasize the major concepts developed throughout the semester.

Students should be prepared to:

- interpret and evaluate class designs;
- explain object lifetime and ownership;
- reason about copying and moving;
- use and interpret overloaded operators;
- compare composition and inheritance;
- reason about polymorphism;
- work with templates and generic programming; and
- evaluate alternative software designs.

The examination will emphasize understanding and application rather than memorization of isolated definitions.

GitHub Portfolio

Students will maintain a GitHub repository containing their work from the course.

The portfolio should demonstrate both completed programming work and professional organization.

Course repositories may include:

- source files;
- header files;
- README files;
- documentation;
- sample input and output;
- test files;
- project materials; and
- other files required by assignments.

The GitHub portfolio will be evaluated separately as part of the final course grade.

---

6. Grading

Grade Distribution

Category	Weight	
Exams (3)	40%	Weeks 4,8,12
Quizzes	20%	Every Friday

Category	Weight	
Final Examination	15%	Monday the 7th of Dec
Programming Project	10%	
GitHub Portfolio	10%	Resume builder
Participation	5%	Just show up
Total	**100%**	

Grade Scale

Grade	Percentage
A	90–100
B	80–89
C	70–79
D	60–69
F	Below 60

Grading Notes

Grades will be based on demonstrated understanding of course concepts as well as successful completion of assigned work.

Programming work may be evaluated for:

- correctness;
- design;
- readability;
- documentation;
- appropriate use of C++ features;
- completeness; and
- the student's ability to explain the submitted solution.

Programming assignments, quizzes, and examinations may build on material introduced in earlier assignments.

---

7. Common Course Policies

Participation

This course depends on discussion, experimentation, troubleshooting, and asking questions.

Participation therefore includes more than simply being physically present in class.

Examples of meaningful participation include:

- attending and participating in class;
- asking questions;
- contributing to class discussions;
- participating in the course communication platform;
- helping identify unclear instructions or unexpected behavior;
- discussing programming and technical concepts;
- helping classmates with conceptual problems; and
- sharing useful discoveries or resources with the class.

Questions are especially valuable. A question asked by one student frequently identifies something that should be clarified for the entire class.

Course Delivery

Course topics may not always be presented in a fixed sequence.

Instruction may be adjusted based on:

- class progress;
- student questions;
- programming assignments and projects;
- current developments in computing; and
- topics that arise naturally while solving problems.

Core course objectives will remain consistent regardless of the exact order in which material is presented.

Assignment and Submission Requirements

Students are responsible for understanding and being able to explain all submitted work.

Unless otherwise stated, assignments should:

- satisfy the stated requirements;
- contain all required files;
- be appropriately organized;
- include documentation when appropriate; and
- be submitted through the required course mechanism.

Students are responsible for confirming that submitted files and repository contents are complete.

Program Execution and Documentation Requirements

Programs must compile or execute successfully.

Programming submissions should:

- follow assignment requirements;
- be reasonably formatted;

- contain meaningful names;
- include appropriate comments;
- include required input or configuration files;
- contain sufficient instructions to run the program; and
- include documentation appropriate to the size of the assignment.

Programs containing syntax errors or programs that cannot be executed may be returned without grading or may receive substantial deductions.

GitHub and Repository Requirements

Students may be required to maintain a GitHub repository containing course work.

Repositories should be organized, readable, and reproducible.

Passwords, private keys, API keys, tokens, credentials, or other sensitive information must not be committed to GitHub.

Students are responsible for verifying that work has been successfully pushed to the required repository.

Understanding and Oral Defense of Submitted Work

Students may be asked to explain, modify, or extend submitted programming assignments.

Questions may concern:

- the purpose of particular code;
- algorithms or data structures used;
- design decisions;
- debugging decisions;
- external resources used; and
- alternative approaches to solving the problem.

Inability to explain submitted work may result in deductions and may result in further review for academic misconduct.

Students should expect programming assignments to provide material for quizzes and examinations.

Use of Large Language Models and AI-Assisted Tools

Large language models and AI-assisted programming tools may be useful for:

- explanation;
- brainstorming;
- debugging;
- documentation;
- code review;
- learning unfamiliar syntax; and
- limited implementation assistance.

These tools should support learning rather than replace it.

Students remain responsible for:

- understanding submitted work;
- verifying generated information and code;
- testing generated solutions;
- complying with assignment-specific restrictions;
- identifying external assistance when required; and
- being able to explain and modify submitted work.

Submitting generated work that a student does not understand does not demonstrate mastery of course material.

Individual assignments may impose additional restrictions or requirements concerning AI-assisted tools.

Attendance

Regular attendance is strongly encouraged and attendance may be recorded.

Students who accumulate excessive absences while performing poorly may be reported through appropriate university procedures.

Classes will not normally be streamed or individually recreated for absent students.

Students are responsible for announcements, material, assignments, and deadlines missed because of an absence.

Classroom Conduct

Students are expected to maintain a classroom environment conducive to learning.

Questions and relevant discussion are encouraged.

Electronic devices may be used when appropriate to course activities. Devices used in a distracting or disruptive manner may be required to be put away.

Students should avoid activities that interfere with their own learning or the learning of others.

Computer and Internet Requirements

Students must have reasonable access to a computer and the Internet to complete course work and access online materials.

Personal computer problems generally do not constitute grounds for extending assignment deadlines.

Students should:

- maintain backups of important work;
- push work to repositories regularly;
- avoid waiting until immediately before a deadline to submit work; and
- notify the instructor promptly when significant technical problems occur.

Campus computing resources are available when personal equipment is unavailable.

Late Work

Late work may be accepted unless an assignment specifies that late submissions are not allowed.

Unless otherwise stated:

- the initial late penalty is **15 points**;
- an additional **5 points** may be deducted for each subsequent class period;
- penalties may continue until the assignment has lost up to 50 points; and
- extremely late work may be refused at the instructor's discretion.

Some assignments, particularly assignments near the end of the semester, may have firm deadlines and may not be accepted late.

Missed Exams and Quizzes

Planned Excused Absences

Students should contact the instructor in advance. An examination may be taken early or another arrangement may be made when appropriate.

Unplanned Documented Absences

The instructor may provide an alternative assessment or use another course assessment, such as the final examination, when appropriate.

Unplanned Undocumented Absences

A missed examination or quiz may receive a grade of zero.

Specific arrangements remain at the instructor's discretion and are subject to department and university policies.

Final Examination Scheduling

The final examination will be administered at the officially scheduled university date and time.

Students should plan travel, employment, weddings, vacations, airline reservations, and other personal activities around the university final-examination schedule.

The final examination will not normally be administered early for personal travel or scheduling convenience.

Exceptions require circumstances recognized through appropriate university procedures.

Testing Procedures

Unless otherwise announced:

- bags, books, electronic devices, and other materials will be stored in an area designated by the instructor;
- only permitted testing materials may remain at the student's desk;

- phones, watches, headphones, and other electronic devices must be turned off and stored; and
- a student who leaves the examination room may be required to submit the examination before leaving.

Additional requirements may be announced for individual examinations.

Academic Collaboration

Students are encouraged to discuss course concepts and help one another understand course material.

Collaboration becomes academic misconduct when a student submits another person's work as their own or provides work for another student to submit.

Students should distinguish among:

- discussing a problem;
- explaining a concept;
- helping locate an error;
- sharing a small example where permitted; and
- providing a solution that another student submits.

Allowing another student to copy submitted work may constitute collusion and may result in academic misconduct penalties for all students involved.

Recording of Classes

Students must obtain permission before recording classroom lectures or activities unless an approved accommodation specifically provides otherwise.

Authorized recordings are intended for the personal educational use of the student making the recording.

Recordings, course materials, slides, assignments, examinations, or other instructor-provided materials may not be publicly distributed or posted online without permission.

---

8. Department Resources and Policies

Computer Science Tutoring

Computer Science tutoring is available in **Bolin Room 119** and through the **Office of Tutoring and Academic Support Programs (TASP)** in Moffett Library.

Tutors may assist students with:

- course concepts;
- programming techniques;
- debugging strategies; and
- homework or project requirements.

Tutors may not complete assignments for students.

Students are encouraged to seek assistance early.

Department Testing Policy

The Department of Computer Science has adopted procedures governing examinations.

Unless otherwise announced by the instructor:

- bags, purses, electronics, books, and other materials will be placed in an area designated by the instructor;
- only permitted testing materials may remain at the student's desk; and
- a student who leaves the examination room may be required to submit the examination and may not be allowed to resume it.

Department Programming Assignment Policy

Students are responsible for understanding programming assignments submitted for credit.

Tests and quizzes may include material from out-of-class programming assignments.

Students may be asked to orally answer questions concerning submitted assignments. Failure to demonstrate an appropriate understanding of submitted work may result in deductions from the assignment grade.

Department Academic Misconduct Procedures

Cheating, collusion, plagiarism, and other forms of academic misconduct are subject to Department of Computer Science and university policies.

Department policy may impose penalties including:

- a non-replaceable zero on an assignment, project, or examination;
- reduction of the final course grade;
- failure of the course for repeated offenses; and
- reporting of incidents to appropriate department and university officials.

Allowing another student to copy submitted work may constitute collusion and academic misconduct.

Students should consult the current MSU Student Handbook and Department of Computer Science policies for complete definitions and procedures.

---

9. University Policies and Resources

Academic Misconduct

University policies concerning cheating, collusion, plagiarism, and other forms of academic misconduct apply to all course work.

Students are responsible for reviewing the current **MSU Student Handbook, Appendix E: Academic Misconduct Policy & Procedures**.

https://msutexas.edu/student-life/_assets/files/handbook.pdf

Students Requiring Accommodations

Students who require academic accommodations because of a disability should contact the appropriate university disability-support office and provide required documentation.

Instructors cannot normally provide formal accommodations until appropriate verification has been received.

The Disability Support Office is located in Clark Student Center 168.

Phone: (940) 397-4140

Midterm Progress Reports

Midterm progress information will be provided through the student's MSU Portal according to university procedures.

Midterm grades are intended to help students evaluate their progress.

They are not reported on the permanent transcript and are not included in the cumulative GPA.

Students earning below a C at midterm are strongly encouraged to:

1. meet with the instructor; and
2. make use of available tutoring resources.

Campus Carry

Texas law and university policy permit licensed handgun holders to carry concealed handguns in permitted campus locations.

Open carry and other prohibited forms of firearm possession remain subject to university and state restrictions.

Students should consult the current MSU Campus Carry Policy for complete requirements and restrictions:

https://msutexas.edu/campus-carry/rules-policies

Tobacco Policy

Midwestern State University prohibits the use of tobacco products in university facilities and other locations covered by university policy.

The policy applies to students, faculty, staff, vendors, guests, and visitors.

Students should consult current university policy for complete requirements.

Moffett Library

Moffett Library provides academic resources and services including:

- books;
- journals;
- research databases;
- multimedia materials;
- study spaces;
- media equipment;
- research assistance; and
- remote access to electronic resources.

Students are encouraged to use library resources for course assignments, research, and technical work.

Student Technical Support

Students experiencing difficulty accessing university technology or online services should use the university's student technical-support resources.

Students should notify the instructor promptly when a university system problem affects course work.

Tutoring and Academic Support

Students are encouraged to make use of university tutoring and academic-support programs.

Seeking assistance early is strongly preferable to waiting until immediately before an examination or assignment deadline.