Programming Assignment 2

Fall 2025 - CPSC 298 C++

Topics Covered:

This assignment will cover the following topics (be sure to review):

- Writing, compiling, and running a C++ program
- Arrays
- Pointers
- Functions
- Classes and Objects

Instructions:

For this assignment we will be creating a simple database. This database will store:

Students
Courses

Each of these represents a **separate class within your code** with their own member variables and functions. **See below for more details on each class.**

The user should be prompted with the following menu and choices to pick from:

1. Add a Student
2. Add a Course
3. Enroll a Student in a Course
4. Drop a Student from a Course
5. List All Faculty for Student
6. Remove a Course
7. Print Student Information
8. Print Course Roster
9. Quit

Repeatedly print this prompt until the user chooses to quit. Make sure each option performs the intended operations, taking in additional user input when necessary. I HIGHLY recommend you use separate functions for each option.

Student

A Student will contain the following members:

Variables: m_name, m_ID, m_numCourses, m_courses

Functions: getName(), getID(), getCourses(), enroll(), drop(), print()

***NOTE: you must also include constructors and destructors as well as any necessary parameters for any of the functions.**

m_name will store the student's name.
m_ID will store their student ID.
m_numCourses will store the number of courses the Student is currently enrolled in.
m_courses will store an array of the courses they are currently enrolled in. Assume a student **cannot enroll in more than 5 courses at a time**.

getName() — returns the name of the student.

getID() — returns the ID of the student.

getCourses() — return the array of the courses the student is currently enrolled in.

enroll() — given a course, add it to the m_courses array for this Student if it is available.

drop() — given a course, remove it from m_courses.

print() — print the Student's name, ID, and list of courses.

Course

A Course will contain the following members:

Variables: m_courseName, m_instructor, m_capacity, m_numStudents, m_students

Functions: getName(), getInstructor(), getNumberStudents(), getStudents(), addStudent()

***NOTE: you must also include constructors and destructors as well as any necessary parameters for any of the functions.**

m_courseName will store the name of the course.
m_instructor will store the name of the instructor.
m_capacity will store the maximum number of students allowed.
m_numStudents will store the number of students currently enrolled.
m_students will store the Students currently enrolled in the course. The number of students **cannot exceed m_capacity**.

getName() — returns the name of the course (m_courseName).

getInstructor() — returns the name of the instructor.

isAvailable() — returns 1 if there is space in the class. Returns 0 if the class is full.

getNumStudents() — returns the number of Students currently enrolled.

getStudents() — returns an array of the Students enrolled in this Course.

addStudent() — adds a Student to the array of enrolled students and returns true (1) if the class is NOT at capacity. If the class is at capacity, the Student is not added and it returns false (0).

Menu Options

Remember that the main program/executing code is a **database**! This means that main() will be responsible for prompting the user for an option, executing said option, and storing all Students and Courses. To keep things simple, assume you will only be working with at most a total of 10 Students and 10 Courses at any given time.

1. **Add a Student**

   Creates a new Student object to be stored in the database. Prompt the user to enter in values for the Student's name and ID.

2. **Add a Course**

   Creates a new Course object to be stored in the database. Prompt the user to enter in values for the Course's name, the name of the instructor, and capacity.

3. **Enroll a Student in a Course**

   Prompt the user for the ID of an EXISTING Student and the name of an EXISTING Course. Enroll the Student in the Course. This change should be reflected in **BOTH** the Student object (m_numCourses, m_courses) and Course object (m_numStudents, m_students).

4. **Drop a Student from a Course**

   Prompt the user for the ID of an EXISTING Student and the name of an EXISTING Course. Drop the Student from the Course. This change should be reflected in **BOTH** the Student object (m_numCourses, m_courses) and Course object (m_numStudents, m_students).

5. **List All Faculty for Student**

   Prompt the user for the ID of an EXISTING Student. Enroll the Student in the Course. List the names of every instructor they are enrolled with.

6. **Remove a Course**

   Prompt the user for the name of an EXISTING Course. Remove this Course from all of the enrolled Students' schedule/list of Courses and then remove it from the database.

7. **Print Student Information**

   Prompt the user for the ID of an EXISTING Student. Use the Student print() function.

8. **Print Course Roster**

   Prompt the user for the name of an EXISTING Course. Use the Student print() function to print EACH of the enrolled Students' information.

9. **Quit**

   Terminate the program.

Some Helpful Tips

- Since we want information to be consistent across multiple objects and multiple scopes, it would be smart to use pointers to avoid accidentally failing to update all copies of each object. By using pointers to these objects within our classes and our database, a change in one location will be reflected in all others which simplifies our work.

- Don't forget to include the class name in front of function names belonging to a specific class when writing the function definition. For example, `std::string Course::getName() {}`

- test Test TEST. This assignment will go much more smoothly if you test each function IMMEDIATELY after you write it. It will save you HOURS of debugging when compared to writing all the code and trying to debug at the end.

- For this assignment, I am not explicitly requiring you to write header files and separate implementation files. However, it would be a good practice of your skills to attempt to make a Student.h, Student.cpp, Course.h, and Course.cpp. There will be a future assignment in which you will be required to make separate files so it may be worth it to give it a shot with this assignment.

README and Comments:

Part of your grade is also determined by the documentation and readability of your code. It is important to include comments throughout your code. You do not need to comment on every single line, however, you should use comments to explain sections or groupings of code at the very least.

You are also required to submit a README file with your code files. I will provide a template on canvas if you wish to use it (I highly recommend downloading it). Your README must include the following sections: (1) Name and description of the project, (2) your Identifying Information, (3) a list of the Source (code) Files being submitted, (4) References you used for help and inspiration (See Collaboration Policy), (5) Known Errors, (6) Build Instructions, and (7) Execution Instructions.

Collaboration Policy:

You are encouraged to work with your peers. Checking each other's work often leads to more robust code for both parties. However, everything you turn in must be YOUR OWN WORK. In addition, make sure to throw them a bone and mention them in your README file in your References section.

The use of AI falls under somewhat similar rules. AI can be a helpful tool but has also been shown to be detrimental to critical thinking skills in some cases [1][2]. In addition, it has a tendency to logic its way to any conclusion, true or false [1]. For these reasons, consulting AI should be a last resort and done incrementally. If you have searched and failed to find a satisfactory solution to a specific problem on your own, AI can be helpful for (1) recommending sources addressing this problem, then (2) explaining the concept of the problem/solution, and then (3) providing the tools/building blocks needed to solve the problem. When using AI, you should ALWAYS start at (1) and move up only if necessary. (3) should not be the go-to. You should ***NEVER*** use AI to write your code for you or represent it as your own work. If you use AI as a reference, include it in the README file and indicate what it was used for.

Lastly, provide website titles or links you used as references. You only need to include those that actually contributed to the development of your code. You do NOT need to include references for course materials (i.e. canvas, slides, or ZyBooks).

Deliverables:

You should submit to canvas by the deadline:

- All source files (.cpp files)
  - main.cpp
- README file

DO NOT submit your executable files.

Due Date:

The current due date is:

Sunday, November 2 (10/24/25)

This is subject to change. I recommend starting early and reviewing the Late Policy on the Syllabus in case you are concerned by this deadline.

Grading Criteria:

	Condition		Automatic Total Grade
1.	No or minimal submission	→	0%
2.	It doesn't compile	→	50%
3.	It doesn't run	→	50%
4.	It runs but it never works	→	60%

If your program at least sometimes works, the remaining points are decided by functionality (how well and consistent it works) and documentation (comments and README). Functionality is an additional 25% of the total grade, and documentation is an additional 15% of the total grade.