

OBJECT ORIENTED PROGRAMMING

10 September 2026

1. Write a C++ program that accepts two integers and implements three separate functions to swap them using:
 - (a) Pass by value
 - (b) Pass by reference
 - (c) Pass by pointer

Demonstrate the result of each function and explain through the program output why the three approaches behave differently.

2. Write a C++ program that dynamically allocates an array of n integers using a pointer. The program should read n , accept all elements from the user, calculate the sum, average, minimum, and maximum, display the results, and release the dynamically allocated memory correctly. Do not use `vector`.
3. Create a C++ program to manage student records using dynamic memory allocation. Each student should contain a roll number, name, and marks. The program should dynamically create n student objects, accept their information, and display the student who has the highest marks. Ensure that all dynamically allocated memory is properly released.
4. Implement a C++ class `FileHandler` that demonstrates RAII. The constructor should open a file, a member function `writeData()` should write text to the file, and the destructor should automatically close the file. Demonstrate the use of the class in a block and show that the file is automatically closed when the object goes out of scope.
5. Create a C++ class `DynamicArray` that dynamically allocates an integer array in its constructor and releases the memory in its destructor. Implement the following functions:

```
insert()
display()
findMaximum()
findMinimum()
```

Do not call `delete[]` from `main()`. Memory cleanup must be handled by the class using RAII.

6. Create a C++ program for an employee management system. Define an **Employee** class containing employee ID, name, and salary. Maintain a collection of employees using `unique_ptr`. Implement functions to add an employee, display all employees, search for an employee by ID, and remove an employee by ID. Do not use raw `new` or `delete`; use `make_unique()`.
7. Create a C++ program containing the classes **Student** and **Course**. A course can have multiple students, and a student can enroll in multiple courses. Use `shared_ptr` to represent shared ownership. Implement functions to add students, add courses, and display the students enrolled in each course and the courses taken by each student.
8. Implement a banking system in C++ containing the classes **Bank**, **Customer**, and **BankAccount**. A customer can have multiple accounts. Create customers and accounts dynamically using smart pointers. Implement operations for depositing money, withdrawing money, displaying account details, and searching for an account. Use `unique_ptr` wherever there is a single owner.
9. Implement a C++ program containing two classes, **PersonA** and **PersonB**, where each person can have a reference to the other as a friend. First implement the relationship using `shared_ptr`. Then modify the program so that one side uses `weak_ptr`. Demonstrate why using `shared_ptr` on both sides can prevent objects from being destroyed and how `weak_ptr` solves the problem.
10. Create a C++ library management system containing the classes **Library**, **Book**, and **Member**. A library contains multiple books, a member can borrow multiple books, and books can exist independently of a particular borrowing relationship. Implement the following operations:

```
addBook()
registerMember()
borrowBook()
returnBook()
displayAvailableBooks()
```

Use appropriate smart pointers to represent ownership and relationships.

11. Design and implement a C++ university management system containing the classes **University**, **Department**, and **Professor**. A university contains multiple departments, a department has multiple professors, a department cannot exist independently of the university, and a professor can exist independently of a department. Implement operations to add departments, add professors, assign professors to departments, and display

department and professor information. Use appropriate smart pointers to represent the relationships.

12. Design and implement a C++ company project management system containing the classes **Company**, **Employee**, and **Project**. A company has multiple employees, an employee can work on multiple projects, and a project can have multiple employees. Employees and projects can exist independently of each other. Implement operations to add employees, create projects, assign employees to projects, and display project and employee information. Use appropriate C++ ownership and relationship mechanisms.
13. Design and implement a C++ e-commerce system containing the classes **Customer**, **Order**, and **Product**. A customer can place multiple orders, an order contains multiple products, an order cannot exist after its customer is removed, and products exist independently of orders. A product can belong to multiple orders. Implement operations to add customers, add products, create orders, add products to orders, and display customer orders and order details.
14. Implement a C++ class **ResourceManager** that manages dynamically allocated **Resource** objects. The class must use RAII and **unique_ptr** for resource ownership. Implement the following functions:

```
addResource()  
removeResource()  
findResource()  
displayResources()
```

Do not use raw **new** or **delete** outside the class.

15. Design and implement a C++ document-sharing system containing the classes **Document**, **User**, and **Editor**. A document can be accessed by multiple users simultaneously. Use **shared_ptr** for shared ownership and **weak_ptr** for non-owning references where appropriate. Implement operations to create a document, open a document, close a document, and display its contents. Demonstrate what happens when the last owner of a document releases it.
16. Design and implement a complete C++ library system containing the classes **Library**, **Book**, and **Member**. The library owns its books, members can borrow books, and multiple members must not own the same book. A book may be temporarily referenced by a member. Use appropriate smart pointers and RAII so that no explicit **delete** appears in **main()**. Implement:

```
addBook(),registerMember(),borrowBook(),returnBook(),searchBook(),  
displayBooks(),displayMemberBooks()
```