University of Central Punjab

**Faculty of Information Technology**

**Object Oriented Programming**

**Spring 2025**

**Lab 05**

# Topic

Classes in C++, access specifiers, objects, methods, constructors, destructors, Passing objects as arguments, Returning Objects, Constructor Initializer List, Static and Const Members, Constant and Static Objects

# Objective

Making students familiarize with the concepts of constructors, destructors, Use of Static, const Members in class and Constant and Static Objects

# CLO

Apply object-oriented programming principles to implement real-world problems.

**Instructions:**

* Indent your code.
* Comment your code.
* Use meaningful variable names.
* Plan your code carefully on a piece of paper before you implement it.
* Name of the program should be same as the task name. i.e. the first program should be Task\_1.cpp
* **void main() is not allowed. Use int main()**
* **You are not allowed to use any built-in functions**
* **You are required to follow the naming conventions as follow:**
* **Variables:** firstName; (no underscores allowed)
* **Function:** getName(); (no underscores allowed)
* **ClassName:** BankAccount (no underscores allowed)

**Students are required to complete the following tasks in lab timings.**

**Task 1: Rectangle Class**

**Problem Statement:**

Create a class named Rectangle that represents a rectangle with the following private attributes:

• length (double)

• width (double)

The class should include the following member functions:

**• Setter Functions:**

• void setLength(double l) – Sets the length of the rectangle.

• void setWidth(double w) – Sets the width of the rectangle.

**• Getter Functions:**

• double getLength() – Returns the length of the rectangle.

• double getWidth() – Returns the width of the rectangle.

**• Constructor:**

• A default constructor that initializes length and width to 0.0.

• A parameterized constructor that accepts the length and width as arguments.

• A Copy Constructor

**• Member Functions:**

• double area() – Calculates and returns the area of the rectangle.

• double perimeter() – Calculates and returns the perimeter of the rectangle.

• Rectangle combine(Rectangle r) – Takes another Rectangle object as an argument and returns a new Rectangle object that represents the union of both rectangles' areas (i.e., the combined rectangle having width and length equal to the maximum width and length of the two).

**• Destructor:**

• A simple destructor.

**Additional Instructions:**

• Write a program that creates two Rectangle objects with different dimensions, combines them, and displays the combined rectangle's area and perimeter.

**Sample Output:**

Rectangle 1: Length = 5.0, Width = 3.0

Rectangle 2: Length = 8.0, Width = 4.0

Combined Rectangle: Length = 8.0, Width = 4.0

Area of Combined Rectangle: 32.0

Perimeter of Combined Rectangle: 24.0

**Task 2: Product Class**

**Problem Statement:**

Create a class named Product that represents a product in an inventory system with the following private attributes:

• productName (char\*)

• price (double)

• quantity (int)

The class should include the following member functions:

**• Setter Functions:**

• void setProductName(char\* name) – Sets the product name dynamically using pointers.

• void setPrice(double p) – Sets the price of the product.

• void setQuantity(int q) – Sets the stock quantity.

**• Getter Functions:**

• char\* getProductName() – Returns the product name.

• double getPrice() – Returns the price.

• int getQuantity() – Returns the stock quantity.

**• Constructor:**

• A default constructor that initializes productName to nullptr, price to 0.0, and quantity to 0.

• A parameterized constructor that accepts const char\* name, double price, int quantity.

• A Copy Constructor that performs a deep copy for char\*.

**• Member Functions:**

• double totalValue() – Calculates and returns the total stock value as price \* quantity.

• void restock (int extra) – This function adds new incoming stock (extra) to the current quantity. Whenever new stock arrives, this function is used to update the existing quantity

• Product mergeProduct(Product p) – Takes another Product object as an argument and returns a new Product object that:

• Has a combined name (productName = Name1 & Name2),

• The maximum price from both products,

• The total quantity as the sum of both products.

**• Destructor:**

• A simple destructor that releases memory allocated to productName and prints: "Product removed from inventory."

**Additional Instructions:**

• Write a program that creates two Product objects with different details, merges them into a new product, and displays the merged product’s name, price and quantity.

**Sample Output:**

Product 1: Name = Pen, Price = 10.0, Quantity = 50

Product 2: Name = Marker, Price = 20.0, Quantity = 30

Merged Product: Name = Pen&Marker

Price = 20.0

Quantity = 80

**Task 3: Student Class**

**Problem Statement:**

Create a class named Student with the following private attributes:

• name (char\*)

• rollNumber (int)

• marks (double)

The class should include the following member functions:

**• Setter Functions:**

• void setName(char\* n) – Sets the name of the student.

• void setRollNumber(int r) – Sets the roll number of the student.

• void setMarks(double m) – Sets the marks obtained by the student.

**• Getter Functions:**

• char\* getName() – Returns the name of the student.

• int getRollNumber() – Returns the roll number of the student.

• double getMarks() – Returns the marks obtained by the student.

**• Constructor:**

• A default constructor that initializes name to "Unknown", rollNumber to 0, and marks to 0.0.

• A parameterized constructor that accepts name, rollNumber, and marks as arguments.

• A Copy Constructor.

**• Member Functions:**

• Student compareMarks(Student s) – Takes another Student object as an argument and returns the Student object with higher marks.

• void displayStudent() – Displays the details of the student.

**• Destructor:**

• A simple destructor.

**Additional Instructions:**

• Write a program to create two Student objects and compare their marks. Display the student with the higher marks.

**Sample Output:**

Student 1:

Name: John Doe

Roll Number: 101

Marks: 85.5

Student 2:

Name: Jane Smith

Roll Number: 102

Marks: 90.0

Student with Higher Marks:

Name: Jane Smith

Roll Number: 102

Marks: 90.0

**Task4** Managing Employee Information with Static Count and Const Values

**Objective:**

Design a class called Employee that have the following attributes:

* char\* name (name of the employee)
* int age (age of the employee)
* float salary (salary of the employee)
* static int employeeCount (static member to count the number of employee objects created)
* const int MAX\_SALARY (constant member variable for the maximum salary allowed) **Required Functions:**
* **Default Constructor**:
* Initialize name to "Unknown", age to 0, and salary to 3000.0.
* Set MAX\_SALARY to 10000 and increment employeeCount.
* **Parameterized Constructor**:
* Initialize name, age, and salary using the constructor initializer list.
* Ensure the salary does not exceed MAX\_SALARY.
* **Destructor**:
* Deallocate dynamic memory for name and decrement the static employeeCount.
* **Setter and Getter functions**:
* Implement setters for name, age, and salary.
* Implement getters to retrieve name, age, and salary. (const functions)
* **Static Function**:
* Implement a static function getEmployeeCount() to return the total count of Employee objects.
* **Main Function**:
* Create multiple Employee objects and display the total count of employees using getEmployeeCount().

SAMPLE OUTPUT

|  |
| --- |
| //After creating 2 Employee objects  Employee e1("John", 28, 9500);  Employee e2("Alice", 30, 11000); // Salary will be set to 10000 due to MAX\_SALARY constraint  // Display total number of employees cout << "Total Employees: " << Employee::getEmployeeCount() << endl;  // Expected Output: Total Employees: 2 // Creating a const Employee object  const Employee e3("Constant Employee", 35, 5000); cout << "Employee 3 Name: " << e3.getName() << endl; cout << "Employee 3 Salary: " << e3.getSalary() << endl;    // Since e3 is a const object, it cannot be modified, and the following would cause an error:  // e3.setSalary(6000); // Uncommenting this would result in a compile-time error    // Expected Output for the const object:  Employee 3 Name: Constant Employee  Employee 3 Salary: 5000 |
|  |

**Task5: Constant Object and Static Data in Student Class** **Objective:**

* Design a Student class that have the following attributes

* char\* name (name of the student)
* int rollNumber (roll number of the student)
* static int totalStudents (static variable to track the total number of students created)
* const int MAX\_ROLL\_NUMBER (constant maximum roll number) **Required Functions:**
* **Default Constructor**:
* Initialize name to "Unknown", rollNumber to 0, and totalStudents to

0.

* Set MAX\_ROLL\_NUMBER to 5000.
* **Parameterized Constructor**:
* Initialize name and rollNumber using the constructor initializer list.
* Increment totalStudents.
* **Destructor**:
* Deallocate memory for name and decrement totalStudents.
* **Static Function**:
* Implement a static function getTotalStudents() that returns the totalStudents.
* **Const Object**:
* In main(), create a const Student object and demonstrate that its attributes cannot be modified after creation.

SAMPLE OUTPUT

|  |
| --- |
| // Creating a Student object using parameterized constructor Student s1("Alice", 101); cout << "Total Students: " << Student::getTotalStudents() << endl; // Expected Output: Total Students: 1    // Creating a const Student object const Student s2("Bob", 202);    // Since s2 is a const object, it cannot be modified, and the following would cause an error:  // s2.setName("John"); // Uncommenting this would result in a compile-time error    cout << "Student 2 Name: " << s2.getName() << endl; cout << "Student 2 Roll Number: " << s2.getRollNumber() << endl; // Expected Output for the const object:  Student 2 Name: Bob  Student 2 Roll Number: 202    cout << "Total Students after creation: " << Student::getTotalStudents() << endl; // Expected Output: Total Students after creation: 2 |
|  |

**Task 6: BankAccount Class with Static Members and Const Variables** **Objective:**

Design a BankAccount class that the following attributes

•

* char\* accountHolderName (name of the account holder)
* int accountNumber (account number)
* double balance (current balance)
* static int totalAccounts (static variable to keep track of total bank accounts created)
* const double MIN\_BALANCE (constant minimum balance

allowed in the account) **Required Functions:**

* **Default Constructor**:
* Set accountHolderName to "Unknown", accountNumber to 0, and balance to 1000.0.
* Set MIN\_BALANCE to 500.0 and increment totalAccounts.
* **Parameterized Constructor**:
* Initialize accountHolderName, accountNumber, and balance using the constructor initializer list.
* If the balance is less than MIN\_BALANCE, set it to MIN\_BALANCE.
* Increment totalAccounts.
* **Destructor**:
* Deallocate memory for accountHolderName and decrement totalAccounts.
* **Setter and Getter Methods**:
* Implement setter and getter functions for accountHolderName, accountNumber, and balance.
* **Static Function**:
* Implement a static function getTotalAccounts() to return the totalAccounts.
* **Main Function**:
* Create a few BankAccount objects and demonstrate how the static variable totalAccounts tracks the total number of created accounts.

SAMPLE OUTPUT

|  |
| --- |
| // Creating 2 BankAccount objects  BankAccount b1("John Doe", 12345, 7000);  BankAccount b2("Jane Smith", 67890, 400); // Will be set to MIN\_BALANCE 500 due to the validation    cout << "Total Bank Accounts: " << BankAccount::getTotalAccounts() << endl; // Expected Output:  Total Bank Accounts: 2    // Displaying account details  cout << "Account Holder Name: " << b1.getAccountHolderName() << ", Balance: " << b1.getBalance() << endl; // Expected Output:  Account Holder Name: John Doe, Balance: 7000    cout << "Account Holder Name: " << b2.getAccountHolderName() << ", Balance: " << b2.getBalance() << endl; // Expected Output:  Account Holder Name: Jane Smith, Balance: 500 (since balance was below  MIN\_BALANCE)    // Creating another BankAccount object  BankAccount b3("Mark Twain", 11223, 10000);  cout << "Total Bank Accounts after 3rd creation: " << BankAccount::getTotalAccounts() << endl;  // Expected Output:  Total Bank Accounts after 3rd creation: 3 |
|  |

## **Task 7: Static Object**

**Objective:**  
Demonstrate how a **static object** retains its state between multiple function calls.

**Problem Statement:**  
Create a class Counter with a static object to persist data across function calls.

**Attributes:**

* int count

**Functions:**

* **Default Constructor:** Initializes count = 0 and prints "Counter initialized."
* **void increment():** Increments count by 1.
* **void displayCount():** Displays current count.
* **Destructor:** Prints "Counter destroyed."

**Sample Code:**

void performTask() {

static Counter c; // Static object

c.increment();

c.displayCount();

}

**Sample Output:**

First call to performTask():

Counter initialized.

Current count: 1

Second call to performTask():

Current count: 2

Third call to performTask():

Current count: 3

Program ending...

Counter destroyed.