LESSON PLAN 2025-26(WINTER)

NAME OF THE TEACHER : DEEPAK KUMAR BARDA, LECT.(STAGE-II,CSE)

Subject: **PROGRAMMING WITH C++(Course Code: CSEPC 201)**
Program: Diploma in Computer Science and Engineering
Semester: 3rd
Total Contact Hours: 45
Total Marks: 100
Assessment: Internal Assessment – 30, End Term – 70

After completion of the course, the students will be able to:
CO1-Describe object-oriented programming (OOP) principles.
CO2-Develop proficiency in C++ syntax and programming constructs.
CO3-Implement advanced OOP features for software design.
CO4-Demonstrate polymorphism and operator overloading.
CO5-Handle exceptions and ensure robust program execution.

Lesson No.	Topic/Sub-Topic	Learning Objective	Activity	Homework	COURSE OBJECTIVE
Introduction to Object-Oriented Programming & C++ Basics(12 Hours)					
1	Introduction to OOP Principles	Students will be able to differentiate between procedural and object-oriented programming paradigms.	Class discussion and whiteboard session contrasting procedural vs. OOP with a real-world analogy (e.g., building a car vs. a house).	Research and write a short summary of the key principles of OOP (Encapsulation, Inheritance, Polymorphism, Abstraction).	CO1
2	User-Defined Types: Structs and Unions	Students will be able to define and access data within struct and union types.	In-class coding: Create a struct to represent a student with name, roll number, and marks.	Write a program that uses a union to store either a student's ID (int) or their name (string) and demonstrates how unions save memory.	CO1
3	Getting Started with C++ Syntax	Students will be able to write and compile a basic "Hello, World!" program.	Live coding demonstration of writing, saving, and compiling a simple C++ program in an IDE.	Write a program that prints your name and a short bio to the console.	CO1
4	Data Types, Variables, and Strings	Students will be able to declare variables with different data types and use the std::string class.	A short quiz on identifying the correct data type for various pieces of information (e.g., an age, a bank balance, a name).	Write a program that asks the user for their favorite movie and year, stores the information in variables, and prints it in a formatted sentence.	CO1

5	Functions and Default Values	Students will be able to define and call functions, and utilize default parameter values.	Pair programming exercise: Each pair writes a function to calculate the area of a rectangle, with default values for length and width.	Create a function to calculate the power of a number, with a default exponent of 2.	CO1
6	Recursion and Function Overloading (Intro)	Students will be able to explain the concept of recursion and recognize its basic implementation.	Group activity: Trace the execution of a recursive factorial function on a whiteboard.	Write a recursive function to calculate the nth Fibonacci number.	CO1
7	Namespaces	Students will be able to use namespaces to organize code and avoid naming conflicts.	Live coding demo showing the problem of name collision and how std:: and using namespace solve it.	Rewrite a simple program to use a custom namespace for your own functions.	CO1
8	C++ Operators	Students will be able to correctly use arithmetic, relational, and logical operators in expressions.	Solve a series of logical and arithmetic expression problems in a timed, in-class quiz.	Write a program that takes two numbers and prints the results of a variety of operations (+, -, *, /, %, ==, >).	CO1
9	Flow Control: If-Else Statements	Students will be able to write conditional statements to control program flow.	"Code completion" exercise: Fill in the blanks of a program that checks if a number is positive, negative, or zero.	Write a program that takes a student's grade as input and prints "Pass" if the grade is 50 or above, and "Fail" otherwise.	CO1
10	Flow Control: Loops	Students will be able to use for, while, and do-while loops to repeat code blocks.	Small-group exercise: Design a program that prints numbers from 1 to 10 using a for loop, a while loop, and a do-while loop.	Write a program that uses a while loop to repeatedly ask the user for input until they enter the word "quit".	CO1
11	Arrays	Students will be able to declare and manipulate single and multi-dimensional arrays.	In-class coding: Create an array of 5 integers and calculate their sum and average.	Write a program that initializes a 2D array (e.g., a tic-tac-toe board) and prints it to the console.	CO1

12	Pointers	Students will be able to use pointers to store memory addresses and dereference them to access values.	Live demo: Draw a diagram on the board illustrating how pointers and variables are stored in memory.	Write a program that declares an integer variable, a pointer to that integer, and prints both the value of the variable and the value accessed through the pointer.	CO1
Unit II: Abstraction Mechanism: Classes & Objects(13 Hours)					
13	Abstraction & Classes (Part 1)	Students will be able to define a basic class and differentiate it from a struct.	Class discussion: Brainstorm the data and behaviors for a Car class.	Define a class for a Rectangle with private member variables for length and width.	CO2
14	Abstraction & Classes (Part 2)	Students will be able to define member data and member functions within a class.	In-class coding: Add a calculateArea() member function to the Rectangle class from the previous lesson.	Add a setLength() and setWidth() member function to the Rectangle class to allow modifying its dimensions.	CO2
15	Public and Private Access	Students will be able to explain and use the public and private access specifiers to enforce encapsulation.	Short group exercise: Analyze a provided code snippet and identify which variables and functions are accessible from outside the class.	Modify the Rectangle class to make the member variables private and provide public "getter" and "setter" methods.	CO2
16	Constructors	Students will be able to create default, parameterized, and copy constructors for a class.	Live coding demo: Create a class with multiple constructors and show how to instantiate objects using each one.	Write a class for a Book with a parameterized constructor that takes title, author, and year as input.	CO2
17	Destructors and Inline Functions	Students will be able to understand the role of destructors and use inline functions to optimize performance.	Walk through a program's execution to demonstrate when a destructor is called.	Create a simple class that allocates memory in its constructor and frees it in its destructor.	CO2

18	Static Members	Students will be able to use static data members and static member functions.	In-class coding: Add a static member variable to a Student class to keep track of the total number of students created.	Write a class with a static member function that returns the total count of objects created.	CO2
19	Friend Functions	Students will be able to declare a friend function to grant access to a class's private members.	Pair programming: Write a function that is not a member of a class but needs to access its private data, then make it a friend.	Create two classes, A and B, and make a function a friend of both to swap their private data members.	CO2
20	References	Students will be able to use references as aliases for variables and as function parameters.	Live coding demo: Show the difference between passing arguments by value, by pointer, and by reference.	Write a function that swaps the values of two integer variables using references.	CO2
21	Single Inheritance	Students will be able to create a derived class that inherits from a single base class.	In-class coding: Create a Vehicle base class and a Car derived class.	Extend the Vehicle and Car example by adding a Bicycle derived class.	CO2
22	Multiple Inheritance	Students will be able to use multiple inheritance to create a class from two or more base classes.	Live coding demo demonstrating the syntax and a potential issue (the "diamond problem") with multiple inheritance.	Define two base classes, Swimmer and Runner, and create a Triathlete class that inherits from both.	CO2
23	Multilevel and Hybrid Inheritance	Students will be able to implement multilevel and hybrid inheritance.	Group exercise: Draw a class hierarchy on the board for a Person -> Employee -> Manager relationship.	Write a program that implements a hybrid inheritance structure.	CO2

24	Virtual Base Class	Students will be able to use a virtual base class to solve the diamond problem.	A step-by-step whiteboard session illustrating the memory layout of an object with and without a virtual base class.	Refactor the code from Lesson 22 to use a virtual base class and demonstrate that the diamond problem is resolved.	CO2
25	Constructors/Destructors in Inheritance	Students will be able to trace the execution order of constructors and destructors in an inheritance hierarchy.	Debugging exercise: Analyze a program with print statements in each constructor and destructor to trace the order of execution.	Create a program with a base class and two levels of derived classes and confirm the constructor/destructor call order.	CO2
Unit III: Inheritance & Polymorphism (Part 1)(7 Hours)					
26	Introduction to Polymorphism	Students will be able to define polymorphism and distinguish between static and dynamic binding.	Class discussion: Use a real-world example like a print() function working on different shapes to explain polymorphism.	Research and write a brief summary on the difference between early binding and late binding.	CO3
27	Static Polymorphism: Function Overloading	Students will be able to overload functions by changing the number or type of their parameters.	In-class coding: Overload a function to calculate the area of both a circle and a rectangle.	Create a function named add that is overloaded to add two integers, two doubles, or three integers.	CO3
28	Dynamic Polymorphism: Base Class Pointer	Students will be able to use a base class pointer to point to a derived class object.	Live coding demo showing how a base class pointer can call a derived class's function if the function is virtual.	Write a simple program with a Shape base class and Circle and Square derived classes, and use a Shape pointer to hold instances of both.	CO3
29	Dynamic Polymorphism: Object Slicing	Students will be able to identify and explain object slicing.	Debugging exercise: Provide a code snippet that demonstrates object slicing and have students identify the bug.	Write a program that purposely causes object slicing and then explain why it occurred.	CO3

30	Late Binding and Virtual Functions	Students will be able to implement late binding using virtual functions for method overriding.	Pair programming: Modify a program to add a virtual keyword to a base class function to enable late binding.	Create a Vehicle class with a virtual drive() function and two derived classes, Car and Motorcycle, that override the function.	CO3
31	Pure Virtual Functions & Abstract Classes (Part 1)	Students will be able to define a pure virtual function and create an abstract class.	Live coding demo: Create a Shape abstract class with a pure virtual getArea() function.	Write a program that defines an abstract base class with a pure virtual function and tries to instantiate an object of that class to see the error.	CO3
32	Pure Virtual Functions & Abstract Classes (Part 2)	Students will be able to create a concrete class by inheriting from and implementing all pure virtual functions of an abstract class.	In-class coding: Create a Circle derived class that inherits from the Shape abstract class and provides an implementation for getArea().	Write a program that creates an abstract class and two concrete derived classes.	CO3
Unit IV: Polymorphism (Part 2) & Operator Overloading(8 Hours)					
33	The this Pointer	Students will be able to explain the purpose of the this pointer and use it in their code.	Live coding demo: Show how to use this to return an object from a member function or to differentiate between member data and a local variable.	Write a class with a setter function that returns a reference to the object (return *this;) to allow for chaining method calls.	CO4
34	Operator Function	Students will be able to write an operator function.	Short quiz on the syntax for operator functions.	Write a class for a Point in a 2D plane and define a member operator function to add two Point objects.	CO4
35	Operator Overloading	Students will be able to overload unary and binary operators for custom classes.	Live coding demo of overloading the ++ and -- operators for a custom class.	Write a class that overloads the + operator to perform vector addition.	CO4
36	Overloading Binary Operators (Part 1)	Students will be able to overload a binary operator for a custom class.	Pair programming: Overload the + operator for a Complex number class.	Extend the Complex number class to also overload the - operator.	CO4

37	Overloading Binary Operators (Part 2)	Students will be able to implement operator overloading as a non-member function.	In-class coding: Rewrite the + operator overload for the Complex class as a non-member function.	Overload the * operator for the Complex class as a non-member function.	CO4
38	Overloading I/O Operators (<< and >>)	Students will be able to overload the stream insertion and extraction operators.	Live coding demo showing how to make a custom class work with cout and cin.	Overload the >> and << operators for the Point class to allow for easy input and output.	CO4
39	More on Operator Overloading	Students will be able to overload the subscript [] and function call () operators.	In-class coding: Overload the [] operator for a custom Array class.	Create a class that overloads the () operator to act as a functor (function object).	CO4
40	Final Review of Operator Overloading	Students will be able to apply the principles of operator overloading to new problems.	Group activity: A code challenge to overload multiple operators for a new class (e.g., a Matrix class).	Debug a provided program with several operator overloads that are not working correctly.	CO4
Unit V: Exception Handling(5 Hours)					
41	Exception Handling: try, throw, and catch	Students will be able to use try, throw, and catch blocks to handle exceptions.	Live coding demo showing how to throw and catch an integer exception.	Write a function that throws an exception if a number is negative and a main function that catches it.	CO5
42	Exceptions and Derived Classes	Students will be able to throw and catch exceptions of derived class types.	In-class coding: Create a base exception class and a derived exception class, then show how a base class catch block can handle both.	Modify the program from Lesson 41 to use a custom exception class.	CO5
43	Function Exception Declaration	Students will be able to declare which exceptions a function can throw.	Class discussion on the benefits and drawbacks of using exception specifications.	Write a function with an exception specification that throws an exception not listed in the specification to see what happens.	CO5

44	Unexpected Exceptions	Students will be able to handle unexpected exceptions using std::unexpected.	Live coding demonstration of a function that throws an exception that is not in its exception specification and is not caught by main.	Research and write a short paragraph explaining the difference between std::bad_exception and std::unexpected.	CO5
45	Comprehensive Review & Q&A	Students will be able to apply all concepts from the course to a comprehensive problem.	Group-based "programming contest" where students must solve a problem using multiple OOP principles and exception handling.	Take-home quiz or a more extensive problem set covering all five units.	CO5

Deepak Kumar Barch
11.07.2025
Signature of Teacher

Deepak Kumar Barh
11.07.2025
Signature of HOD