DATA STRUCTURES
30251201/34251201

COURSE OBJECTIVES

- To be familiar with the use of data structures as the foundational base for computer solutions to problems.
- To understand various techniques of searching and sorting.
- To understand basic concepts about stacks, queues, lists, trees and graphs.

Unit I:

Introduction to Data Structures: Algorithms & their Characteristics, Asymptotic Notations, **Array:** Representations of Array, Index to Address Translation, **Link List:** Introduction, Implementation of Linked List, Operations, Circular Link List, Doubly Linked List, Polynomial Manipulation using Linked List.

Unit II:

Stacks: Concepts and Implementation of Stacks, Operations on Stack, Conversion of Infix to Postfix Notation, Evaluation of Postfix Expression, Recursion.

Queues: Concepts and Implementation, Operations on Queues, Dequeue, Priority Queues, Circular Queue.

Unit III:

Trees: Types, Terminology, Binary Tree -Representations, Traversal, Threaded Binary Tree, Binary Search Tree, Height Balanced Tree-AVL Tree.

Graph: Terminologies, Representation of Graphs- Sequential & Linked Representation, Graph Traversals- BFS, DFS, Spanning Trees.

Unit IV:

Searching: Linear Search, Jump Search, Binary Search, Hashing and Collision Resolution Techniques; **Sorting:** Bubble Sort, Selection Sort, Insertion Sort.

Unit V: (Dynamic Content)

Graph-Based Data Structures: k-d Trees, R-Trees, **Dynamic/Self-Adjusting Data Structures: Splay Trees, Hashing Techniques for Databases & Storage**: Extensible Hashing, Cuckoo Hashing, Perfect Hashing.

RECOMMENDED BOOKS

1. Data Structures, Algorithms and Applications in C++, Sartaj Sahni, 2nd Edition.
2. An Introduction to Data Structures with Applications, Jean-Paul Tremblay, Mcgraw hill.
3. Data Structures & Algorithms, Aho, Hopcroft & Ullman, Original Edition, Pearson Publication.

COURSE OUTCOMES

After completion of this course, the students would be able to:

CO1. explain the concept of algorithms, linked list data structures.

CO2. apply the appropriate data structure stack and queue to solve problems.

CO3. discuss the concept of tree and graph data structure, types & their applications.

CO4. design various searching and sorting algorithms and analyze their performance.

CO5. discover the applications of data structure in emerging areas and real world.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	1	1	1				1	1		2	2	1
CO2	3	3	2	1	1				1	1		2	3	2
CO3	3	2	2	1	1	2			1	1		2	2	3
CO4	3	3	2	2	2	1	1	1	2	2	2	2	3	1
CO5	2	2	3	2	3	2	1	1	2	2	2	3	2	3

OBJECT ORIENTED PROGRAMMING
30251202/34251202

COURSE OBJECTIVES

- To study about the concept of object-oriented programming.
- To create C++ programs that leverage the object-oriented features.
- To apply object-oriented techniques to solve real world problems.

Unit I:

Object Oriented Paradigm, Features of OOPs: Encapsulation, Class and Object, Inheritance, Reusability, Polymorphism, Abstraction etc, Comparison with Procedural Oriented Programming & Object-Oriented Programming, Function Overloading, Default Augments, References, Inline Functions.

Unit II:

Classes & Objects: Specification of Class, Visibility Modes, Defining Member Functions, Creating of Objects, Static Data Member, Static Member Function, Array of Objects, Object as Arguments, Friend Function and Class, Member Function, Member Initializer List, Constructors and Destructors, Difference between Class and Structure.

Unit III:

Dynamic Allocations: New, Delete, Malloc and Free, Dynamic Allocation of Objects, Array of Objects, Mutable Data Members, Self-Referential Class, Shallow and Deep Copying, This Pointer.

Operator Overloading: Overloading Unary and Binary Operators, Type Casting.

Unit IV:

Inheritance: Introduction to Code Reuse, Visibility Modes, Types of Inheritance, Ambiguity in Inheritance, Virtual Base Classes, Constructors in Derived Classes.
Polymorphism: Dynamic and Static Binding, Pure Virtual Function, Abstract and Concrete Classes, Virtual Destructors, Containership: Nesting of Classes.

Exception Handling: Try, Catch and Throw, Streams and File: Basic Concept and Class Hierarchy.

Unit V: (Dynamic Content)

OOPs in Current Technologies and Languages: Java, React js, C#, and JavaScript, Kotlin and Swift, **Design Principles and Patterns**: SOLID Principles, Object Cloning and Metaprogramming, Dynamic OOP in Web Frameworks, etc.

RECOMMENDED BOOKS

1. C++ How to Program: H M Deitel and P J Deitel, Prentice Hall,1998.
2. Object Oriented Programming in Turbo C++: Robert Lafore, The WAITE Group Press, 1994.
3. Programming with C++: D Ravichandran, T.M.H, 2003.
4. Object oriented Programming with C++: E Balagurusamy, Tata McGraw-Hill, 2001.
5. The Complete Reference in C++: Herbert Schildt, TMH, 2002.
6. Object Oriented Analysis & Design: G. Booch, Addison Wesley, 2006.
7. Principles of Object-Oriented Analysis and Design: James Martin, Prentice Hall, 1992.

COURSE OUTCOMES

After completion of this course, the students would be able to:

CO1. understand Fundamental concepts and features of Object-Oriented Programming such as classes, objects, encapsulation, inheritance, polymorphism, and abstraction.

CO2. apply OOP constructs including constructors, destructors, static members, friend functions, operator overloading, and dynamic memory allocation to develop modular programs.

CO3. discuss different types of inheritance, visibility modes, binding mechanisms, and object-copying techniques to determine their suitability in object-oriented design.

CO4. demonstrate concept of polymorphism, abstract classes, exception handling, and file handling to create robust and reusable OOP-based applications.

CO5. design & develop modern OOP concepts, principles and contemporary OOP tools to scalable simple applications in current technologies.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	3	1	1	2			1	1	2	1	2	3	1
CO2	3	3	2	1	2			1	1	2	1	2	3	1
CO3	3	3	2	1	3			1	1	2	1	2	3	1
CO4	3	3	3	2	3			1	1	2	1	2	3	1
CO5	3	2	3	2	3	1	1	1	1	2	1	2	3	1

DISCRETE STRUCTURES
30251203/34251203

COURSE OBJECTIVES

- To gain knowledge of basic algebra and discrete numeric function.
- To learn functions and its relation.
- To familiarize with propositional logic.
- To know about the graph theory and its application in computer.
- To be familiar with generating function.

Unit I:

Functions and Relations: Sets, Subsets, Power Sets, Complement, Union and Intersection, Demorgan's Law Cartesian Products, Relations, Relational Matrices, Properties of Relations, Equivalence Relation, Functions, Injection, Surjection and Bijective Mapping, Composition of Functions, Permutations, Characteristic Functions and Mathematical Induction.

Unit II:

Partial Order Relations and Lattice: Partial Order Set, Hasse Diagrams, Upper Bounds, Lower Bounds, Maximal and Minimal Element, First and Last Element, Lattices, Sub-Lattices, Lattice Homomorphism, Lattice Isomorphism, Complete Lattice, Complemented Lattice, Distribution Lattice.

Unit III:

Group and Field: Group axioms, Abelian group, and its properties, Sub group, Co-sets, Left and Right Co-sets, Normal subgroup, semi group, Lagrange theorem, fields, minimal polynomials, reducible polynomials, primitive polynomial, polynomial roots, applications.

Unit IV:

Graph Theory: Finite Graphs, Incidence and Degree, Isomorphism, Sub Graphs and Union of Graphs, Connectedness, Walk, Paths and Circuits, Eulerian and Hamiltonian Graphs. Trees: Properties of Trees, Pendant Vertices in Tree, Center of Tree, Spanning Trees and Cut Vertices, Binary Tree, Matrix Representation of Graph, Incidence and Adjacency Matrix and Properties, Applications of Graphs in Computer Science.

Unit V: (Dynamic Content)

Discrete Numeric Functions: Introduction to Discrete Numeric Functions and Generating Functions, Introduction to Recurrence Relations, Solution of Combinatorial Problem Using Generating Functions, Linear Recurrence Relations with Constant Coefficients, Homogeneous Solutions, Particular Solutions, Total Solutions, Solution of Linear Recurrence Relations using Method of Generating Functions.

RECOMMENDED BOOKS

1. Discrete Mathematical Structures with Application to Computer science: J.P Tremblay and Manohar, McGraw-Hill, 1st Edition 2017.
2. Graph Theory: Nersingh Deo, PHI Learning, 2014.
3. Discrete Mathematics: C.L Liu, 4th Edition 2012.
4. Discrete Mathematics and its Applications: Rosen, McGraw Higher Ed, 7th Edition 2008.
5. Topics in Algebra: N. Herstein, Wiley, 2nd Edition 2006.

COURSE OUTCOMES

After completion of this course, the students would be able to:

CO1. acquire knowledge of set theory.

CO2. analyse the concept of Lattices.

CO3. apply the concept of Group Theory.

CO4. derive the Inferences from Graph theory.

CO5. illustrate the Discrete numeric function and recursive relation.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	1	1		1	1			1			2	1
CO2	3	3	2	2	1				1	2		1	3	2
CO3	3	3	2	2	1			1		2		1	3	2
CO4	3	3	3	2	2	1			1	3	1	2	3	3
CO5	3	2	3	2	2				1	2	1	2	3	2

PROBABILITY AND RANDOM PROCESSES
30251204/34251204

COURSE OBJECTIVES

- To learn central tendency, skewness and kurtosis.
- To describe probability theory and distribution.
- To familiarize with correlation and regression.
- To know about the hypothesis analysis.
- To explore the theory of attributes and rules of association.

Unit I:

Measure of Central Tendency: Measures of Averages and Standard Deviation, Moments About Origin and Mean, Moment Generating Function, Skewness and Kurtosis.

Unit II:

Probability & Regression: Definition of Probability: Classical and Axiomatic Approaches, Laws of Total and Compound Probability, Conditional Probability, Curve Fitting, Correlation and Regression.

Unit III:

Probability Distribution: Probability Distribution Function, Probability Density Function, Central Limit Theorem, Binomial Distribution, Poisson Distribution, Normal Distribution, Exponential Distribution, Uniform Distribution.

Unit IV:

Testing of Hypothesis: Testing of Hypothesis, Chi-Squre Test, T-Test, F-Test, Z-Test, Analysis of Variance: One Way and Two-Way Classifications.

Unit V: (Dynamic Content)

Random Variables & Processes: Concept of Random Variable, One-Dimensional Random Variable, Two-Dimensional Random Variable, Distribution Function, Joint Probability Distribution Function, Marginal Probability Distribution, Differences between Marginal, Joint and Conditional distributions, Cumulative Probability Distribution, Conditional Distribution Function, Graphical interpretation of Conditional CDF.

RECOMMENDED BOOKS

1. Mathematical Statistics: M Ray and H.S. Sharma, Ram Prasad Publications, 3rd Edition 2017.
2. Statistical Methods: V. K. Kapoor, S.C. Gupta, S. Chand& Company, 11th Edition 2018.
3. Probability: T. Veerarajan, Statistics and Random Processes, McGraw Hill, 3rd Edition 2008.
4. Introduction to Probability Models: S. M. Rose, Elsevier, 10th Edition 2011.

COURSE OUTCOMES

After completion of this course, the students would be able to:

CO1. gain knowledge of measures of central tendency.

CO2. evaluate the skewness, kurtosis, curve fitting, correlation and regression.

CO3. interpret the theory of probability and its distributions.

CO4. select and justify appropriate hypothesis-testing techniques for real-world problems and interpret statistical evidence for decision-making.

CO5. compute random variables with random process.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	1	1						1	1	2	1	1
CO2	3	3	2	2	1				1	2	1	2	2	1
CO3	3	3	2	2	1	1		1		2	2	2	3	2
CO4	2	3	2	2	2	1	1		1	1	2	3	2	3
CO5	3	2	2	2	2				1	2	2	3	3	2

BASIC ELECTRICAL & ELECTRONICS ENGINEERING
30251205/34251205

COURSE OBJECTIVES

- To impart basic knowledge of the DC and AC circuits and their applications.
- To familiarize the students with the basic knowledge of magnetic circuits, transformer, rotating electrical machine and its terminology.
- To make familiarize the students about the working of, various electronic circuits and its importance.

Unit I:

D.C. Circuits Analysis: Voltage and Current Sources, Dependent and Independent Source, Source Conversion, Kirchhoff's Law, Mesh and Nodal Analysis, Network Theorems: Superposition Theorem, Thevenin's Theorem & Norton's Theorem and their Applications.

Unit II:

Single-Phase AC Circuits: Generation of Sinusoidal AC Voltage, Definitions: Average Value, R.M.S. Value, Form Factor and Peak Factor of AC Quantity, Concept of Phasor, Analysis of R-L, R-C, R-L-C Series and Parallel Circuit, Power and Importance of Power Factor, Resonance in AC Circuits.

Unit III:

Transformer & Electrical Machines: Magnetic Circuits and Electromagnetism, Transformers: Construction, principle, types, losses & efficiency, OC & SC Test DC Machines: Motor and Generator working Principles, Characteristics, Introduction to Induction Motors and Synchronous Machines.

Unit IV:

Digital Electronics, Devices & Circuits: Number Systems, Logic Gates and Truth Tables, Diodes, Transistors (BJT), Multiplexers, Demultiplexers.

Unit V: (Dynamic Content)

Emerging Trends and Applications: Introduction to Smart Grids, Smart Meters, and Renewable Systems. Types of earthing, Sensors and Basic IoT Applications.

RECOMMENDED BOOKS

1. Basic Electrical and Electronics Engineering, D.P. Kothari and I.J. Nagrath, 2nd Edition, McGraw-Hill Education, 2020.
2. Basic Electrical and Electronics Engineering, S.K. Bhattacharya, 2nd Edition, Pearson Education, 2017.
3. Basic Electrical Engineering, V.N. Mittle and Arvind Mittal, 2nd Edition, McGraw-Hill Education, 2005.
4. Basic Electrical Engineering, A.E. Fitzgerald, David E. Higginbotham, and Arvin Grabel, 5th Edition, McGraw-Hill Education, 1981.
5. Principles of Electrical Engineering and Electronics, V.K. Mehta and Rohit Mehta, Revised Edition, S. Chand Publishing, 2019.

COURSE OUTCOMES

After completion of this course, the students would be able to:

CO1. apply fundamental laws and network theorems to analyze DC circuits

CO2. analyze single-phase series & parallel AC circuits for calculation of power, power factor, and resonance conditions.

CO3. explain the working principles, construction, and operational characteristics of transformers, DC machines, and induction motors.

CO4. design basic digital logic circuits using logic gates, flip-flops, and counters

CO5. discuss the concepts of smart meter, smart grids, earthing, and IoT systems to emerging industrial applications in automation and renewable energy systems.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	3	2	2	1							1	2	2
CO2	3	3	2	2	1							1	2	3
CO3	3	2	3	2	2	1						2	2	2
CO4	3	3	3	2	1			1	2	2		1	3	3
CO5	3	2	3	2	3	2	2	2		1	1	2	2	2

SUSTAINABILITY & ENVIRONMENTAL SCIENCE
30251211/34251211

COURSE OBJECTIVES

- To equip students with a comprehensive understanding of environmental science, pollution control, sustainability, and global frameworks, enabling them to analyze environmental challenges and contribute to sustainable solutions through informed decision-making and responsible practices.

Unit I:

Introduction to Environmental Science: Definition, Importance and its Components. Ecosystem and its Components, Water Cycle, Carbon Cycle, Food Chain, Energy Flow in Ecosystem, Current State of Environment in India and World: Underlying Reasons (Root Causes) of Modern Environmental Degradation (Social, Psychological, Cultural).

Unit II:

Environmental Pollution and Management: Air, Water, Noise, Soil, Thermal and Radioactive, Causes, Impacts, Pollution Control Techniques and Mitigation Strategies, Solid Waste Management: Principles of Waste Management, Different Components of Waste Management System and Introduction to Management of Hazardous Waste Like E-Waste, Plastic Waste, Global Environmental Issues: Climate Change, Global Warming, Ozone Layer Depletion.

Unit III:

Environmental Policies and Laws in India: Environmental Protection Act, Water Act, Air Act, **Overview of Global Environmental Policies and Frameworks**: Kyoto Protocol, Montreal Protocol, COP Summits. Introduction to Clean Development Mechanism, Carbon Credit, Carbon Trading.

Unit IV:

Sustainability Concepts: Definition, Importance, Pillars of Sustainability (Economic, Environmental, and Social), Sustainable Development, Overview of UN Sustainable Development Goals (SDGS) And Their Global Relevance, Concept of Circular Economy, Resource Efficiency, Energy Conservation, Green Buildings and Sustainable Manufacturing.

Unit V:

Sustainable Energy Solutions: New Energy Sources: Need of New Sources, Different Types New Energy Sources, Applications of Hydrogen Energy, Ocean Energy Resources, Tidal Energy Conversion, Concept, Origin and Power Plants of Geothermal Energy, Introduction to Sustainable Transportation Systems and Sustainable Water Infrastructure.

RECOMMENDED BOOKS

1. A Text Book of Environmental Studies, D. K. Asthana, Meera Asthana, S Chand & Co., New Delhi.
2. Environmental Engineering & Management, S. K. Dhameja, S K Kataria & Sons, New Delhi
3. Environmental Pollution Control Engineering, C.S. Rao, New Age International Publishers
4. Environmental Sustainability and Green Technologies, A. K. Gupta, PHI Learning.

COURSE OUTCOMES

After completion of this course, the students would be able to:

CO1. explain the fundamental concepts of environmental science, including ecosystems and the causes of environmental degradation.

CO2. analyze the sources, causes, and impacts of air, water, and solid waste pollution and propose appropriate mitigation strategies.

CO3. evaluate the effectiveness of environmental policies and global frameworks in addressing environmental challenges.

CO4. explain the concepts of sustainability and sustainable development goals.

CO5. apply various solutions for achieving sustainable development.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	1					2	3	1		1	1	2	1	2
CO2	1	1		1		3	3	1	1	1	2	3	1	2
CO3	1	1	1	1	1	3	3	1	1	1	2	3	1	2
CO4	1					2	3	1		2	1	3	1	2
CO5	1	1	1	2	2	3	3	1	2	2	3	3	2	2

DATA STRUCTURES LAB
30251206/34251206

LIST OF PROGRAMS

1. Write a program to implement doubly linked list with all possible deletion operations.
2. Write a program to insert an element in the beginning of the circular linked list.
3. Write a program to implement stack using linked list.
4. Write a program to count the number of nodes in the binary search tree.
5. Write a program to implement AVL Tree.
6. Write a program to traverse the BST in pre-order and post-order.
7. Write a program to implement Graph using an array.
8. Write a program to implement Breadth First Search.
9. Write a program to implement Depth First Search.
10. Write a program to implement Spanning Tree.
11. Write a program to implement binary search algorithm.
12. Implement a simple Bloom Filter in C++ that supports insertion of string(s) and checks possibly contain string(s) using multiple hash functions.
13. Implement a Count–Min Sketch to process a stream of strings and answer approximate frequency queries.

COURSE OUTCOMES

After completion of the course students would be able to:

CO1. implement data structures like arrays and linked lists to manage and manipulate data effectively.
CO2. develop stack and queue operations, including applications like expression evaluation and recursion.
CO3. construct and traverse various tree structures, including binary search trees and height-balanced trees.
CO4. apply graph traversal techniques like BFS and DFS for solving pathfinding and connectivity problems.
CO5. design efficient sorting and searching algorithms, and resolve hashing collisions.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	3	3	2	3	1		1	3	2	3	2	3	1
CO2	3	3	3	2	3	1		1	3	2	3	2	3	1
CO3	3	3	3	2	3	1		1	3	2	3	2	3	1
CO4	3	3	2	2	3	1	1	1	3	2	3	2	3	1
CO5	3	3	3	2	3	1	1	1	3	2	3	2	3	1

OBJECT ORIENTED PROGRAMMING LAB
30251207/34251207

LIST OF PROGRAMS

1. Write a program to demonstrate example of member initializer list.
2. Write a program to demonstrate example of default constructor or no argument constructor.
3. Write a program to demonstrate example of parameterized constructor.
4. Write a program to demonstrate example of copy constructor.
5. Write a program to demonstrate example of constructor overloading.
6. Write a program to demonstrate example of destructors.
7. Write a program to demonstrate example of constructor using this pointer.
8. Write a program to demonstrate example of constructor with default arguments.
9. Write a program to dynamic Initialization of Objects.
10. Write a program to set values of data members using default, parameterized and copy constructor
11. Write a program to demonstrate example of simple inheritance.
12. Write a program to demonstrate example of private simple inheritance.
13. Write a program to read and print student's information using two classes and simple inheritance.
14. Write a program to demonstrate example of multilevel inheritance.
15. Write a program to read and print employee information using multiple inheritance.
16. Write a program to demonstrate example of multiple inheritance.
17. Write a program to demonstrate example of hierarchical inheritance to get square and cube of a number.
18. Write a program to read and print employee information with department and PF information using hierarchical inheritance.
19. Write a program for unary minus (-) operator overloading.
20. Write a program for unary increment (++) and decrement (--) operator overloading.
21. Write a program for unary logical NOT operator overloading.
22. Write a program to add two objects using binary plus (+) operator overloading.
23. Write a program to add two distances using binary plus (+) operator overloading.
24. Write a program to create a simple class and object.
25. Write a program to create an object of a class and access class attributes.
26. Write a program to create multiple objects of a class.
27. Write a program to create class methods.
28. Write a program to define a class method outside the class definition.
29. Write a program to assign values to the private data members without using constructor.
30. Write a program to create an empty class (a class without data members and member functions).
31. Write a program to create a class with setter and getter methods.
32. Write a program to create a class to read and add two distances.

33. Write a program to create a class for student to get and print details of a student.
34. Write a program to create a class for student to get and print details of N students.
35. Write a program to demonstrate example of array of objects.
36. Write a program to create class to read and add two times.
37. Write a program to create class to read time in seconds and convert into time in (HH:MM:SS).
38. Write a program to create class to read time in HH:MM:SS format and display into seconds.
39. Write a program to demonstrate example of friend function with class.
40. Write a program to count the created objects using static member function.
41. Write a program to create an object of a class inside another class declaration.
42. Write a program to create a class Point having X and Y Axis with getter and setter functions.
43. Write a program for passing an object to a Non-Member function.
44. Write a program for accessing Member Function by pointer.
45. Write a program for accessing the address of an object using 'this' pointer.
46. Write a program to create a class with public data members only.
47. Write a program to input list of candidates and find winner of the Election based on received votes.
48. Write a program for Banking Management System using Class.
49. Write a program to create a file.
50. Write a program to read a text file.
51. Write a program to read and write text in/from file.
52. Write a program to read and write values using variables in/from file.
53. Write a program to Build a simple calculator API using OOP and C++ web framework.
54. Create a class and expose it through a web API.

COURSE OUTCOMES

After completion of the course students would be able to:

CO1. use the concepts of object-oriented programming, including classes, objects, constructors, destructors, and initializer lists.

CO2. apply OOP principles to develop programs using operator overloading and friend functions.

CO3. analyze relationships among classes through inheritance, object interaction, and dynamic initialization.

CO4. demonstrate object-oriented program structure using file handling, static members, and pointers.

CO5. design OOP-based applications, including web-exposed C++ classes and APIs.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	1	1	1	1	1	1		2			2	1
CO2	3	2	2	1	1					2		1	3	2
CO3	3	3	2	2	1					2		1	2	2
CO4	3	3	2	2	1			1		2		2	3	2
CO5	3	3	3	2	2	1	1	1	1	3	2	2	2	3

ELECTRICAL & ELECTRONICS ENGINEERING LAB
30251208/34251208

LIST OF EXPERIMENTS

1. To verify Kirchhoff's Current Law & Kirchhoff's Voltage Law.
2. To verify Superposition Theorem
3. To determine resistance & inductance of a choke coil.
4. To determine active & reactive power in a single-phase A.C circuit.
5. To determine voltage ratio & current ratio of a single-phase transformer.
6. To determine the polarity of a single-phase transformer.
7. To perform open circuit & short circuit test on a single-phase transformer.
8. To study multimeter & measure various electrical quantities
9. To study of constructional details of DC machine.
10. To determine the V-I characteristics of diode in forward bias & reverse bias condition.
11. To determine phase and line quantities in three phase star and delta connection
12. To study of effect of open and short circuits in simple circuits
13. To plot Transistor CB characteristics (Input and Output)
14. To plot Transistor CE characteristics (Input and Output)
15. Study the output characteristics of a solar PV panel under varying conditions
16. Develop a simple IoT system to monitor temperature and humidity using sensors.

COURSE OUTCOMES

After completion of the course students would be able to:

CO1. demonstrate the ability to operate lab equipment and instruments relevant to the electrical engineering field.
CO2. collect experimental data accurately and effectively.
CO3. integrate theoretical knowledge from coursework into practical applications and experiments.
CO4. communicate experimental results effectively through oral presentations and written documentation.
CO5. demonstrate responsibility and professionalism in the completion of lab tasks and assignments.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	2	3	3	3	3	2				2	2	2
CO2	2	3	2	3	2	3	2	3		2		2	2	3
CO3	3	3	3	3	2	2	2	2	2	3	2	3	2	2
CO4	1	2	2	3		2	2	3	3	3	2	2	3	3
CO5			1	2		3	3	3	3	3	2	3	2	2

LIST OF MICRO PROJECT-II
30251210/34251210

1. Library Book Tracker (Stack Implementation)

- **Scenario**: You're working for a library, and your task is to manage the last book returned to the library. Implement a stack to keep track of books, allowing for operations like push (add a book), pop (remove the most recently returned book), and peek (check which book was last returned).

2. Customer Service Call Center (Queue with Two Stacks)

- **Scenario**: You're managing a call center where customer calls come in and wait to be attended. Use two stacks to simulate how calls are processed in the queue, where enqueue adds a call to the waiting line, and dequeue processes the first call in line.

3. Parentheses Validator (Balanced Parentheses Checker)

- **Scenario**: You're building a code editor, and you need to verify that users' code has properly balanced brackets. Implement a stack to scan through the input code and determine if all opening brackets have a matching closing bracket.

4. Text Reverser (Reverse a String Using Stack)

- **Scenario**: You work for a social media app that wants to implement a 'reverse text' feature for creative posts. Use a stack to reverse the text input by a user, flipping it from end to start.

5. Ride Ticket Queue (Circular Queue)

- **Scenario**: You manage the queue system for a popular amusement park ride. Implement a circular queue to ensure that once the end of the line is reached, people move to the start of the line in a loop as needed.

6. Video Playback (Linked List Reversal)

- **Scenario**: You're building a video player app, and the app needs a feature to go backward through previously watched scenes. Implement a function to reverse the linked list of scenes for playback.

7. Palindrome Detector (Palindrome Checker Using Stack)

- **Scenario**: You're developing a game that asks players to identify palindromic words. Implement a function using a stack to check if a given word reads the same forwards and backwards.

8. Emergency Response System (Priority Queue Implementation)

- **Scenario**: In a city, an emergency response system needs to prioritize fire, medical, and police calls. Implement a priority queue where each type of emergency has a priority level, ensuring the most urgent calls are attended first.

9. Survey Analysis (Find Middle of a Linked List)

- **Scenario**: You're analyzing customer feedback, and you need to find the central opinion from a linked list of responses. Implement a function to find the middle response efficiently.

10. Detecting Loop in a Delivery Route (Cycle Detection in Linked List)

- **Scenario**: You're working on route planning software for delivery trucks and need to identify if the route has loops that would cause the driver to visit the same location twice. Use Floyd's Cycle-Finding algorithm to detect loops in the route.

11. Personal Finance App (Binary Search Tree Implementation)

- **Scenario**: You're developing a personal finance app to manage and search for transactions. Create a binary search tree (BST) to store and find transactions based on date, amount, or type.

12. Website Page History (BST Level Order Traversal)

- **Scenario**: You're building a browser feature that allows users to see their page history in order of visits. Implement a level order traversal on a BST that stores pages based on their visit date and time.

13. Friendship Network (Graph Representation)

- **Scenario**: You're designing a social network platform. Represent the network of friends using adjacency lists or an adjacency matrix to store connections between users.

14. Friend Finder (Depth-First Search)

- **Scenario**: You work for a dating app, and users want to find friends of their friends. Implement DFS to traverse the friendship graph and find potential matches.

15. City Transportation Planner (Breadth-First Search)

- **Scenario**: You're working on an app that plans the shortest route between subway stations. Use BFS to traverse the transportation network and find the shortest path from one station to another.

16. Event Merging (Merge Two Sorted Linked Lists)

- **Scenario**: You're developing an event planning app and need to combine two lists of event times while keeping them sorted. Implement a function that merges two sorted linked lists.

17. Online Shopping Cart (Hash Table Implementation)

- **Scenario**: developing an online store and need to keep track of product IDs and their availability. Use a hash table to store and quickly look up items in the shopping cart.

18. Mouse Movement extension

- **Scenario**: This following program makes use of some sub function, which were already discussed previously (Mouse Movement), and shows how they can be used to write useful programs like free-hand drawing. Below are the functions used:
  - **initmouse():** use to initialize mouse.
  - **showmouse():** shows mouse pointer on the output screen.
  - **hidemouse():** used to hide mouse while drawing.
  - **getmouseposition():** Fetches current location of the pointer and draw line accordingly.

19. Missing number in array

- **Scenario**: Given an array of size N-1 such that it only contains distinct integers in the range of 1 to N. Displaymissing element. Complete the function MissingNumber() that takes array and N as input parameters and returns the value of the missing number.
  Input:
  N = 5
  A[] = {1,2,3,5}
  Output: 4

20. Leaders in an Array

- **Scenario**: Given an array A of positive integers. Your task is to find the leaders in the array. An element of array is leader if it is greater than or equal to all the elements to its right side. The rightmost element is always a leader.
  The task is to complete the function leader() which takes array A and n as input parameters and returns an array of leaders in order of their appearance.
  Input:
  n = 6
  A[] = {16,17,4,3,5,2}
  Output: 17 5 2
  Explanation: The first leader is 17 as it is greater than all the elements to its right. Similarly, the next leader is 5. The right most element is always a leader so it is also included.

21. K^{th} Smallest Element

- **Scenario**: Given an array arr[] and an integer K where K is smaller than size of array, the task is to find the Kth smallest element in the given array. It is given that all array elements are distinct.
  Your task is to complete the function k^{th} Smallest() which takes the array arr[], integers l and r denoting the starting and ending index of the array and an integer K as input and returns the K^{th} smallest element.
  Input:
  N = 6
  arr[] = 7 10 4 3 20 15
  K = 3
  Output: 7
  Explanation: 3rd smallest element in the given array is 7.

22. Majority Element

- **Scenario**: Given an array A of N elements. Find the majority element in the array. A majority element in an array A of size N is an element that appears more than N/2 times in the array. The task is to complete the function majority Element () which returns the majority element in the array. If no majority exists, return -1.
  Input:
  N = 5
  A[] = {3,1,3,3,2}
  Output:
  3
  Explanation:
  Since, 3 is present more than N/2 times, so it is the majority element.

23. Minimum Number of Jumps

- **Scenario**: Given an array of N integers arr[] where each element represents the maximum length of the jump that can be made forward from that element. This means if arr[i] = x, then we can jump any distance y such that $y \leq x$.

Find the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then you cannot move through that element.
Note: Return -1 if you can't reach the end of the array.
Your task is to complete function minJumps() which takes the array arr and it's size N as input parameters and returns the minimum number of jumps. If not possible return -1.
Input:
N = 11
arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}
Output: 3
Explanation:
First jump from 1st element to 2nd element with value 3. Now, from here we jump to 5th element with value 9, and from here we will jump to the last.

24. Equilibrium Point

- **Scenario**: Given an array A of n positive numbers. The task is to find the first Equilibrium Point in an array. Equilibrium Point in an array is a position such that the sum of elements before it is equal to the sum of elements after it. The task is to complete the function equilibriumPoint() which takes the array and n as input parameters and returns the point of equilibrium. Return -1 if no such point exists.
  Note: Retun the index of Equilibrium point.
  Input:
  n = 5
  A[] = {1,3,5,2,2}
  Output: 3
  Explanation:
  equilibrium point is at position 3 as elements before it (1+3) = elements after it (2+2).

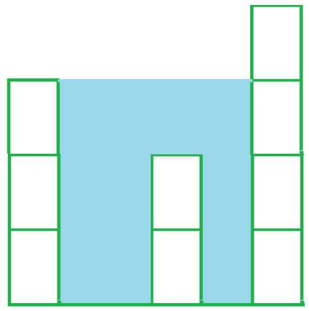
25. Inversion Count in an Array

- **Scenario**: Given an array of integers. Find the Inversion Count in the array. Inversion Count: For an array, inversion count indicates how far (or close) the array is from being sorted. If array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum. Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.
  Your task is to complete the function inversionCount() which takes the array arr[] and the size of the array as inputs and returns the inversion count of the given array.
  Input: N = 5, arr[] = {2, 4, 1, 3, 5}
  Output: 3
  Explanation: The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

26. Trapping Rain Water

- **Scenario**: Given an array arr[] of N non-negative integers representing the height of blocks. If width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.
  Input:
  N = 6

arr[] = {3,0,0,2,0,4}
Output:
10
Explanation:



The task is to complete the function trappingWater() which takes arr[] and N as input parameters and returns the total amount of water that can be trapped.

27. Minimum Platforms

- **Scenario**: Given arrival and departure times of all trains that reach a railway station. Find the minimum number of platforms required for the railway station so that no train is kept waiting. Consider that all the trains arrive on the same day and leave on the same day. Arrival and departure time can never be the same for a train but we can have arrival time of one train equal to departure time of the other. At any given instance of time, same platform can not be used for both departure of a train and arrival of another train. In such cases, we need different platforms.
  Your task is to complete the function findPlatform() which takes the array arr[] (denoting the arrival times), array dep [] (denoting the departure times) and the size of the array as inputs and returns the minimum number of platforms required at the railway station such that no train waits.
  Note: Time intervals are in the 24-hour format (HHMM), where the first two characters represent hour (between 00 to 23) and the last two characters represent minutes (this may be > 59).
  Example 1:
  Input: n = 6
  arr [] = {0900, 0940, 0950, 1100, 1500, 1800}
  dep [] = {0910, 1200, 1120, 1130, 1900, 2000}
  Output: 3
  Explanation:
  Minimum 3 platforms are required to safely arrive and depart all trains.

28. Next Greater Element in an Array

- **Scenario**: Given an array arr[ ] of size N having elements, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.
Input:
N = 4, arr[] = [1 3 2 4]
Output:
3 4 4 -1
Explanation:
In the array, the next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4? since it doesn't exist, it is -1.
You only need to complete the function nextLargerElement() that takes list of integers arr[ ] and N as input parameters and returns list of integers of length N denoting the next greater elements for all the corresponding elements in the input array.

29. Dynamic Memory Allocator Simulation Using Linked Lists

- **Scenario**: Simulate a memory allocation system using linked lists to represent memory blocks.

30. Graph-Based Route Finder

- **Scenario:** Build a route-finding application that uses shortest path algorithms.

31. Efficient Sorting Visualizer

- **Scenario:** Develop an interactive visualization tool for sorting algorithms such as QuickSort, MergeSort, and HeapSort.

32. Event Calendar

- **Scenario:** Use a hash table to store events with quick search functionality by date.

33. Weather Data Analysis

- **Scenario:** Use binary search trees to organize and retrieve temperature records efficiently.

34. Memory Matching Game

- **Scenario:** Develop a matching game that can check if the two flipped cards have the same values and if they have the same "flipped" property (flipped: true), then we can remove them from the list.

35. Employee Management System Using OOP

- **Scenario:** Design a system that stores and manages employee data such as personal information, salary, and work hours. Implement inheritance to classify employees into different categories.

36. Vehicle Rental Service

- **Scenario:** Create classes for vehicles and rentals to manage vehicle availability and bookings. The application manages vehicle availability, customer bookings, rental duration, and cost calculation. This project demonstrates **class design, inheritance, polymorphism, data structures.**

37. Student Quiz System

- **Scenario:** The Student Quiz System is a lightweight software application designed for conducting quizzes, evaluating student responses, and storing results. Students can log in, attempt quizzes, view their marks, and get instant feedback. Teachers can upload

questions, check analytics, and manage quiz records. This project is ideal for C++ backend combined with simple HTML pages for the user interface.

38. QR Code Based Attendance

- **Scenario:** The QR Code Attendance System allows students to mark their attendance by scanning a unique QR code generated by the teacher. This system reduces manual errors, prevents proxy attendance, and maintains secure digital records. This project is ideal for C++ backend combined with simple HTML pages for the user interface.

39. Smart Leave Management System (Students + Faculty)

- **Scenario**: The Smart Leave Management System is a lightweight application that helps students request leave easily and allows faculty/administrators to review, approve, or reject leave applications. It digitalizes the entire leave process and removes paperwork, delays, and manual tracking. The system maintains separate access for students, faculty, and HOD/admin, allowing smooth communication and transparent approval flow. This project is ideal for C++ backend combined with simple HTML pages for the user interface.

40. AI-Assisted Student Feedback & Suggestion System

- **Scenario:** This system allows students to submit feedback about classes, faculty, teaching methods, and campus facilities. The system analyzes feedback and provides sentiment analysis reports (Positive/Neutral/Negative) and automatically suggests improvements to faculty or administration. The AI part can be lightweight—using rule-based analysis or simple keyword scoring implemented in C++.

41. Competitive Coding Smart Coding Tracker

- **Scenario:** A platform where students solve problems daily, update progress, and get difficulty recommendations. Track completed problems by categories (Array, DP, Graphs). Weekly and monthly progress graphs. Badges/achievements for consistency. Priority queues (to recommend problems), graphs for topic linking, hashing for fast lookups.

42. Pair Programming & Code Collaboration Platform

- **Scenario:** This software allows two or more users to write and edit code together in real-time, just like collaborating in Google Docs. Students can learn coding together, debug faster, and work on assignments or competitive programming problems as a team.

43. Smart Timetable & Classroom Monitoring System

- **Scenario:** The Smart Timetable & Classroom Monitoring System is a lightweight software tool that helps colleges or schools manage class schedules, monitor classroom usage, and provide real-time updates to students and faculty. This project is ideal for C++ backend combined with simple HTML pages for the user interface.

44. Code Bracket Validation Tool

- **Scenario:** Develop a tool that checks whether a piece of code contains correctly matched brackets. It should scan through any code snippet and determine whether all types of parentheses are properly opened and closed.

45. Amusement Park Ride Queue Manager

- **Scenario:** Design a simulation for how visitors wait for a popular amusement park ride. Once the line reaches the end, it cycles automatically to the beginning for efficient management. Features: Add a visitor to the queue, remove visitor when ride is completed, Show current line, Continuous circular movement simulation.

46. Video Scene Playback Reversal Module

- **Scenario:** Create a feature for a video player that allows users to watch previously viewed scenes in reverse order. Each scene is stored, and the system plays them back starting from the most recent. Features: Store watched scenes, Reverse the playback order, Display scenes one-by-one, Simple navigation controls.

47. City Emergency Response Dispatcher

- **Scenario:** Create a system that allows city authorities to log incoming emergency calls and automatically categorize them based on their urgency. The most urgent complaint should be attended first. Features: Add new emergency request, assign priority based on emergency type, display next emergency to handle, Emergency logs management.

48. Delivery Route Loop Detection System

- **Scenario:** Develop a system for delivery companies that verifies whether a driver travels through the same location more than once. The system analyzes the route to detect looping paths. Features: Input delivery route, detect presence of route loops, show loop start point if found, Display "No Loop" if path is clean.

49. Address Book Management System

- **Scenario:** This project involves developing a simple Address Book application that enables users to store and manage personal contact information efficiently. The system allows users to add, update, delete, and search contacts by name. Each contact record contains details such as name, phone number, and address. The application uses C++ structures or classes to store data and offers menu-driven navigation for user-friendly operations. This project demonstrates file handling, functions, basic data structures, and console-based data management.

50. Student Report Card Generator

- **Scenario:** This project focuses on creating a Report Card Generator in C++, where the system collects student marks for different subjects, calculates total marks, percentage, and assigns grades automatically. The data is stored using structures, and functions are used to perform operations like input, calculation, and display of the report card. The output is presented in a well-formatted manner, making the project ideal for demonstrating logic building, modular code design, and structure-based data handling.

COURSE OUTCOMES

After completion of the course students would be able to:

CO1. explain fundamental data structures, algorithms, and object-oriented concepts to solve real-world problems.

CO2. apply appropriate data structures, algorithms, and programming techniques to implement efficient solutions in areas such as arrays, stacks, queues, linked lists, trees, graphs, and hash tables.

CO3. analyze and evaluate problem scenarios, compare alternative approaches, and determine optimal solutions for computational and real-life applications.

CO4. design software systems and applications using modular programming, classes, inheritance, and file handling to address practical problems in domains like finance, transportation, social networks, and gaming.

CO5. demonstrate proficiency in implementing advanced algorithms, including searching, sorting, pathfinding, and optimization techniques, and evaluate their performance in terms of correctness and efficiency.

CO-PO Mapping Matrix														
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	1	1	2			1	1	1	2	2	2	2
CO2	3	3	2	2	3			1	1	1	2	2	3	2
CO3	3	3	2	3	3	1		1	2	1	2	3	3	3
CO4	3	2	2	2	3	1	1	1	3	2	3	2	3	3
CO5	3	3	3	2	3	2	1	1	3	2	3	3	3	3