Department of CSE

Details of Lesson Plan

S.No.	Particulars	Details
1.	Course Name	Programming in C
2.	Course Code	BCAMJ-101
3.	Academic Year	2024-25
4.	Semester	1st
5.	Number of Lesson plans	48
6.	Faculty Assigned	Ms. Annu Sonania

Faculty Signature

Lesson Plan No. 1.1	Course Name: Programming in C Topic: Evolution of Programming Languages	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the evolution of programming languages. b. Identify different generations of programming languages. c. Discuss the impact of this evolution on modern programming
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What do you understand by programming languages? - How do you think programming languages have evolved over time? b. Introduction to Evolution of Programming Languages: - Discuss the history and evolution of programming languages from first-generation to current languages. https://www.youtube.com/watch?v=n5Rl9wZJ4Bo **2. Development (30 minutes)** a. Generations of Programming Languages (15 minutes): - Explain the different generations of programming languages (e.g., machine language, assembly language, high-level languages). - Example: Compare COBOL (second-generation) and Python (fourth-generation). - Hands-on: Discuss how different generations of languages impact programming. https://www.youtube.com/watch?v=n5Rl9wZJ4Bo b. Impact on Modern Programming (15 minutes): - Explain how the evolution has led to modern programming practices and languages. - Real-world examples of how evolution has improved programming efficiency. - https://www.youtube.com/watch?v=Kl2KN56XtsY

	3. Exercise (5 minutes) Activity: - Pair students and ask them to research a programming language from a specific generation and present its key features and impact. - Discuss their findings and correct any misconceptions.
Closure	1. Summarize key points: Evolution of programming languages, their generations, and impact on modern programming. 2. Suggested Readings from Textbooks: **BOOK 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 1: "Introduction to Programming" (pp. 1-10) **BOOK 2**: "Programming with C" by Byron Gottfried Chapter 1: "Overview of C Programming" (pp. 1-15)
Evaluation	1. Reflective Questions:How has the evolution of programming languages influenced modern programming practices? 2. Homework: Write a short essay on the evolution of programming languages and their impact on programming efficiency.

Lesson Plan No. 1.2	Course Name: Programming in C Topic: Structured Programming	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the principles of structured programming. b. Apply structured programming concepts to C Programming. c. Differentiate between structured and unstructured programming.
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What do you know about structured programming? - How is it different from unstructured programming? b. Introduction to Structured Programming: - Define structured programming and its principles (e.g., sequence, selection, iteration). https://www.youtube.com/watch?v=9bYj1H4R1Xg **2. Development (30 minutes)** a. Principles of Structured Programming (15 minutes): - Explain the core principles and how they improve code readability and maintainability. - Example: Compare a structured program with an unstructured one. - Hands-on: Write a simple C program using structured programming principles. https://www.youtube.com/watch?v=9bYj1H4R1Xg b. Structured Programming in Python (15 minutes): - Demonstrate how to apply structured programming concepts in C. - Real-world examples of structured programming in C applications. https://www.youtube.com/watch?v=wR9Cww0zVE8 **3. Exercise (5 minutes)** Activity: - Pair students and have them refactor a given unstructured C code into a structured format. - Discuss their answers and correct any misconceptions
Closure	1. Summarize key points: Principles of structured programming and its application in C Programming. 2. Suggested Readings from Textbooks: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy

	Chapter 2: "Structured Programming" (pp. 15-35) **BOOK 2**: "Programming with C" by Byron Gottfried Chapter 2: "Structured Programming Concepts" (pp. 16-30)
Evaluation	1. Reflective Questions:What are the benefits of using structured programming in C Programming? 2. Homework: Write a program using structured programming principles and describe its advantages over unstructured programming

Lesson Plan No. 1.3	Course Name: Programming in C Topic: Compilation Process	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the stages of the compilation process. b. Identify the role of each stage in compiling a C program. c. Describe common compilation errors and how to resolve them
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** **a.** Pre-Discussion Questions: - What do you think happens when you compile a C program? - What are some common errors you've encountered while compiling code? **b.** Introduction to the Compilation Process: - Overview of the stages: Preprocessing, Compilation, Assembly, and Linking. - YouTube link: https://www.youtube.com/watch?v=aTfGxCqP5FA **2. Development (30 minutes)** **a.** Compilation Stages (15 minutes): - Explain each stage in detail with examples. - Hands-on example: Trace the compilation process of a simple C program. https://www.youtube.com/watch?v=kx5STjrVOMI **b.** Common Errors and Troubleshooting (15 minutes): - Discuss common compilation errors and how to resolve them. - Example errors: Syntax errors, missing headers, etc. **3. Exercise (5 minutes) –** Activity: - Provide a C program with intentional errors. Ask students to compile and identify the errors. - Discuss solutions and correct the errors in class.
Closure	1. Summarize key points: Stages of the compilation process and common errors. 2. Suggested Readings from Textbooks: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy Chapter 3: "The Compilation Process" (pp. 21-30) **BOOK 2**: "Programming with C" by Byron Gottfried

	Chapter 3: "Compiling and Debugging" (pp. 31-45)
Evaluation	1. Reflective Questions:How does understanding the compilation process help in debugging programs? 2. Homework:Document the compilation process for a provided C program and list common errors with solutions.

Lesson Plan No. 1.4	Course Name: Programming in C Topic: Object Code, Source Code, Executable Code	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Differentiate between object code, source code, and executable code. b. Understand the role of each type of code in the compilation process. c. Explain how these codes interact in the process of program execution
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** **a.** Pre-Discussion Questions: - What are the different types of code you know of in programming? - How do you think source code is transformed into a running program? **b.** Introduction to Code Types: - Define and explain source code, object code, and executable code **2. Development (30 minutes)** **a.** Code Types (15 minutes): - Describe each type of code with examples. - Show how source code is compiled into object code and linked to form executable code. https://www.youtube.com/watch?v=YZ3m_ZzDgd4 **b.** Interaction and Execution (15 minutes): - Explain how these types of code interact during program execution. - Discuss a real-world example of how source code becomes an executable program.https://www.youtube.com/watch?v=Ivs8i13teGo **3. Exercise (5 minutes)** Activity: - Provide students with source code and ask them to describe the corresponding object code and executable code. - Discuss their observations and clarify any confusion.
Closure	**1.** Summarize key points: Differences and roles of source code, object

	code, and executable code. **2.** Suggested Readings from Textbooks: **BOOK 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 4: "Code Types and Their Roles" (pp. 31-40) **BOOK 2**: "Programming with C" by Byron Gottfried Chapter 4: "From Source Code to Executable" (pp. 46-60)
Evaluation	**1.** Reflective Questions:How do source code, object code, and executable code contribute to the program's execution? **2.** Homework: Write a brief explanation of each type of code and how they work together to run a program.

Lesson Plan No. 1.5	Course Name: Programming in C Topic: Operating Systems	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the basic functions of operating systems. b. Describe the role of operating systems in managing hardware and software resources. c. Identify different types of operating systems and their characteristics.
Teaching Aids (if any)	a. Slides with diagrams and examples a. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What is an operating system? - What functions do you think an operating system performs b. Introduction to Operating Systems: - Overview of the functions and types of operating systems **2. Development (30 minutes)** a. Functions of Operating Systems (15 minutes): - Explain core functions: process management, memory management, file system management, and device management. - Real-world examples of operating systems (Windows, Linux, macOS). https://www.youtube.com/watch?v=Qf1KKIuNVkw b. Types of Operating Systems (15 minutes): - Discuss different types: single-user vs. multi-user, real-time systems, embedded systems. - Compare and contrast different operating systems **3. Exercise (5 minutes) –** Activity: - Ask students to research and present on a specific operating system, covering its functions and characteristics. - Discuss their findings and address any questions. https://www.youtube.com/watch?v=bbf-3H4vVV8
Closure	**1.** Summarize key points: Functions and types of operating systems. **2.** Suggested Readings from Textbooks: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy

	Chapter 5: "Introduction to Operating Systems" (pp. 41-50) **BOOK 2**: “Programming with C” by Byron Gottfried Chapter 5: "Operating System Fundamentals" (pp. 61-75)
Evaluation	1. Reflective Questions**:**How does an operating system manage hardware resources effectively 2. Homework:Write a short essay on the role of operating systems in managing computer resources.

Lesson Plan No. 1.6	Course Name: Programming in C Topic: Fundamentals of Algorithms	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: 1. Understand what an algorithm is and its importance in programming. 2. Identify different types of algorithms and their uses. 3. Develop basic algorithms for common problems.
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What is an algorithm? - Why are algorithms important in programming? b. Introduction to Algorithms: - Define an algorithm and discuss its importance. - Introduce basic types of algorithms (e.g., sorting, searching). **2. Development (30 minutes)** a. Types of Algorithms (15 minutes): - Discuss common algorithms such as bubble sort, linear search, and binary search. - Use flowcharts to illustrate how these algorithms work. https://www.youtube.com/watch?v=4v7dzv7uQm0 b. Algorithm Development (15 minutes): - Hands-on activity: Develop algorithms for simple problems (e.g., finding the maximum number in a list). - Practice converting algorithms into pseudocode. https://www.youtube.com/watch?v=2Imt1ERu83M **3. Exercise (5 minutes) –** Activity: - Provide a problem and ask students to design an algorithm and present it in pseudocode. - Discuss and refine their algorithms in class.
Closure	1. Summarize key points: Importance and types of algorithms, and how to develop them. 2. Suggested Readings from Textbooks: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy

	Chapter 6: "Introduction to Algorithms" (pp. 51-60) **BOOK 2**: “Programming with C” by Byron Gottfried Chapter 6: "Algorithm Fundamentals" (pp. 76-90)
Evaluation	1. Reflective Questions:How do algorithms contribute to solving programming problems efficiently? 2. Assignment:Write an algorithm for a common task and convert it into pseudocode.

Lesson Plan No. 1.7	Course Name: Programming in C Topic: Flow Charts and Pseudocode	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the purpose of flowcharts and pseudocode in programming. b. Create flowcharts to represent algorithms. c. Develop pseudocode for simple algorithms
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What are flowcharts and pseudocode? - How can they help in programming? b. Introduction to Flowcharts and Pseudocode: - Define flowcharts and pseudocode and their purposes. - Explain basic flowchart symbols and pseudocode conventions. **2. Development (30 minutes)** a. Creating Flowcharts (15 minutes): - Demonstrate how to create a flowchart for a simple algorithm. - Hands-on activity: Draw a flowchart for a basic task (e.g., calculating the factorial of a number). https://www.youtube.com/watch?v=2gFntJMRkXY b. Developing Pseudocode (15 minutes): - Show how to write pseudocode for the same algorithm. - Hands-on activity: Convert a flowchart into pseudocode. https://www.youtube.com/watch?v=F_mhL4G1JkU **3. Exercise (5 minutes) –** Activity: - Provide a problem and ask students to create both a flowchart and pseudocode for it. - Discuss and compare their solutions.

Closure	1. Summarize key points: Purpose and creation of flowcharts and pseudocode. 2. Suggested Readings from Textbooks: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy Chapter 7: "Flowcharts and Pseudocode" (pp. 61-70) **BOOK 2**: "Programming with C" by Byron Gottfried Chapter 7: "Design Tools: Flowcharts and Pseudocode" (pp.

	91-105)
Evaluation	1. Reflective Questions:How do flowcharts and pseudocode assist in programming and debugging? 2. Homework:Create flowcharts and pseudocode for a given programming problem.

Lesson Plan No. 1.8	Course Name: Programming in C Topic: Introduction to Compilation and Linking	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the compilation and linking process in C programming. b. Describe the role of object files and libraries. c. Explain how linking integrates different pieces of code into a final executable.
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What is linking in programming? - How does linking differ from compilation? b. Introduction to Compilation and Linking: - Overview of the compilation and linking process. **2. Development (30 minutes)** a. Compilation and Linking Process (15 minutes): - Explain how source code is compiled into object files and how linking combines these object files into an executable. - Illustrate with an example involving multiple C files. https://www.youtube.com/watch?v=bhWJcV54cYw b. Object Files and Libraries (15 minutes): - Describe the role of object files and static/dynamic libraries. - Show how libraries are linked to a program. https://www.youtube.com/watch?v=DJGQk71xH7U **3. Exercise (5 minutes) –** Activity: - Provide a simple project with multiple C files. Ask students to compile and link it, then identify and explain the resulting files (object files, executable). - Discuss the process and any issues encountered.
Closure	1. Summarize key points: Compilation, linking, object files, and libraries. 2. Suggested Readings from Textbooks: **BOOK 1**: “Programming in ANSI C” by E. Balagurusamy Chapter 8: "Compilation and Linking" (pp. 71-80) **BOOK 2**: “Programming with C” by Byron Gottfried

	Chapter 8: "Linking and Libraries" (pp. 106-120)
Evaluation	1. Reflective Questions:How does the linking process impact the final executable? 2. Homework:Compile and link a multi-file C program, then explain each step of the process.

Lesson Plan No. 2.1	Course Name: Programming in C Topic: Programming in C	Course No.: COM-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the significance and structure of C programming. b. Write and execute simple C programs. c. Identify and use basic C language constructs including data types, operators, and expressions. d. Explain key concepts like keywords, identifiers, constants, and variables.
Teaching Aids (if any)	a. Slides with sample code and diagrams b. Video clips demonstrating basic C programming concepts c. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What programming languages are you familiar with? - Why do you think learning C is important? - Can you name a few applications or systems that use C? **b. Programming in C:** - Define C programming language: "C is a high-level programming language that is widely used for system and application development." - Briefly discuss the importance of C in the development of modern software and systems. **Video Reference: "Programming in C"** https://www.youtube.com/watch?v=KJgsSFOSQv0 **2. Development (30 minutes)** **a. Importance of C (5 minutes):** - Overview of C's influence on other languages (e.g., C++, Java). - Use cases: Operating systems, embedded systems, and application development. **Video Reference:** "Why Learn C Programming?" https://www.youtube.com/watch?v=0lFvGu2yXzY

	b. Basic Structure of C Programs (10 minutes): - Explain the basic structure: `#include`, `main()`, `{}`, `return 0;` - Write a simple "Hello, World!" program on the board. **Video Reference:** "C Programming Basics" https://www.youtube.com/watch?v=8PopJ8qclhI **c. Executing a C Program (5 minutes):** - Describe the process: writing code, compiling with a compiler (e.g., GCC), and running the executable. **Video Reference:** "How to Compile and Run a C Program" https://www.youtube.com/watch?v=H2aRGtTaUgI **d. Character Set, Keywords, Identifiers (10 minutes):** - **Character Set:** Describe valid characters in C (letters, digits, and special characters). - **Keywords:** Explain reserved words like `int`, `return`, `void`. - **Identifiers:** Define naming conventions for variables, functions, etc. **Video Reference:** "Understanding C Keywords and Identifiers" https://www.youtube.com/watch?v=7Rt4ZCzA9RU **3. Exercise (5 minutes)** - **Activity:** Have students write a simple C program that uses basic data types, variables, and prints a message. - **Example Task:** Create a program that declares a variable, assigns it a value, and prints it.
Closure	1. Summarize key points: - The importance of learning C. - Basic structure of a C program and how to execute it. - Understanding C's character set, keywords, and identifiers. **2. Suggested Readings** **Book 1:"Programming in ANSI C"** by E. Balagurusamy **Chapter 1:** Programming in C, pp. 1-20 **Book 2:"Programming with C"** by Byron Gottfried **Chapter 1**: Programming in C, pp. 1-25

Evaluation	1. **Reflective Questions:** Discuss why C is still relevant and how understanding its basics can aid in learning other programming languages. 2. **Practice Question:** Write a simple C program that demonstrates the use of variables, operators, and basic I/O functions.

Lesson Plan No. 2.2	Course Name: Programming in C Topic: Understanding Data Types and Variables	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Identify and understand different data types in C programming. b. Explain the role and importance of variables in C programming. c. Differentiate between primitive and complex data types. d. Apply data types and variables effectively in C programs.
Teaching Aids (if any)	a. Slides with diagrams and examples b. Video clips demonstrating data types and variables c. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What data types are you familiar with in other programming languages? - Why do you think data types and variables are crucial in programming? - Can you think of examples where different data types are used? **b. Introduction to Data Types and Variables:** - Define data types: "Data types in C specify the type of data a variable can hold, such as integers, floating-point numbers, and characters." - Discuss the importance of selecting appropriate data types and variables for effective memory management and accurate data representation. ***Video Reference:*** **Understanding Data Types in C Programming** **2. Development (30 minutes)** **a. Basic Data Types (10 minutes):** - *Overview:* Introduce basic data types like `int`, `float`, `char`, and `double`. *Examples:* Discuss how each data type is used and its

	typical applications. ***Video Reference:*** **Primitive Data Types in C** **b. Variables (10 minutes):** - *Definition:* Explain what variables are and how they are used to store data. - *Naming Conventions:* Discuss rules for naming variables and the significance of meaningful names. ***Video Reference:*** **Variables in C Programming** **c. Complex Data Types (5 minutes):** - *Overview:* Introduce complex data types such as arrays, structures, and pointers. - *Examples:* Explain how these types are used to manage collections of data and more complex data structures. ***Video Reference***: Understanding Arrays and Structures in C **3. Exercise (5 minutes)** - ***Activity:*** Have students write a simple C program that declares various types of variables, assigns values to them, and prints these values. - ***Example Task:*** Create a program that declares an `int`, a `float`, and a `char` variable, assigns values, and prints them.
Closure	1. Summarize key points: - The significance of understanding different data types and how they influence memory usage and data manipulation. - The role of variables in storing and managing data. - Recap the types of data and variables discussed and their applications in C programming. **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 2: Data Types and Constants, pp. 25-46 Chapter 3: Variables and Storage Classes, pp. 47-68 **Book 2: "Programming with C" by Byron Gottfried** Chapter 3: Data Types and Variables, pp. 35-52 Chapter 4: Operators and Expressions, pp. 53-72

Evaluation	1. **Reflective Questions:** Discuss how choosing appropriate data types and variables can impact the efficiency and clarity of a program. 2. **Practice Question:** Write a C program that demonstrates the use of different data types and variables, and explains why each type was chosen.

Lesson Plan No. 2.3	Course Name: Programming in C Topic: Character Set, Keywords, Identifiers	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the character set used in C programming. b. Identify and use C keywords effectively. c. Define and apply identifiers in C programming.
Teaching Aids (if any)	a. Slides with diagrams and examples b. Video clips demonstrating character sets, keywords, and identifiers. c. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What characters are used in programming languages you've worked with? - Why is it important to understand reserved keywords in a language? - How do identifiers help in programming? **b. Introduction to Character Set, Keywords, and Identifiers:** - Define character set: "The character set in C programming includes letters, digits, and special characters that can be used in code." - Explain keywords: Reserved words in C with specific meanings. - Discuss identifiers: Names used for variables, functions, etc. **Video Reference**: Understanding C Keywords and Identifiers **2. Development (30 minutes)** **a. Character Set (10 minutes):** - Overview: Explain valid characters in C (letters, digits, special characters). - Examples: Show examples of valid and invalid characters in C code. **Video Reference**: Character Set in C Programming **b. Keywords (10 minutes):** - Overview: Discuss common C keywords like int, return, void,

	etc. - Examples: Illustrate how keywords are used in C programs. **Video Reference**: C Programming Keywords **c. Identifiers (10 minutes):** - Definition: Explain the rules for naming identifiers and their significance. - Examples: Discuss examples of good and bad identifiers. **Video Reference**: Identifiers in C Programming **3. Exercise (5 minutes)** - Activity: Have students create a list of valid and invalid identifiers and keywords in C. - Example Task: Write a simple program using different keywords and identifiers.
Closure	**1.** Summarize key points: - Importance of understanding the character set, keywords, and identifiers in C programming. - How these elements contribute to writing correct and effective C code. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 1: Programming in C, pp. 1-20 **Book 2:** "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie Chapter 2: Types, Operators, and Expressions, pp. 35-57
Evaluation	**1.** Reflective Questions: How do keywords and identifiers affect code readability and maintenance? **2.** Practice Question: Write a program that demonstrates proper use of identifiers and keywords.

Lesson Plan No. 2.4	Course Name: Programming in C Topic: Constants and Variables	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Define and use constants and variables in C programming. b. Understand the differences between constants and variables. c. Apply constants and variables effectively in C programs.
Teaching Aids (if any)	a. Slides with sample code and diagrams b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is the difference between a constant and a variable? - Can you give examples of when to use constants versus variables? **b. Introduction to Constants and Variables:** - Define constants and variables: "Constants are values that do not change, while variables can store data that may change during program execution." - Discuss the role and importance of each in programming. **Video Reference**: Constants and Variables in C Programming **2. Development (30 minutes)** **a. Constants (10 minutes):** - Definition: Explain how constants are defined using const and #define. - Examples: Show examples of using constants in C programs. **Video Reference**: Using Constants in C **b. Variables (10 minutes):** - Definition: Explain variable declaration, initialization, and usage. - Examples: Illustrate examples of variables and how they store and manipulate data.

	Video Reference: Understanding Variables in C **c. Practical Application (5 minutes):** - Overview: Discuss scenarios where constants and variables are used. - Examples: Demonstrate the use of constants and variables in practical programming tasks. **Video Reference**: Variables and Constants in Practice **3. Exercise (5 minutes)** - Activity: Have students write a program that uses both constants and variables. - Example Task: Create a program that declares a constant for a fixed value and a variable for a user-input value.
Closure	**1. Summarize key points:** - Importance of using constants and variables correctly in C programming. - Practical applications and examples of constants and variables. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 2: Data Types and Constants, pp. 25-46 **Book 2:** "Programming with C" by Byron Gottfried Chapter 3: Data Types and Variables, pp. 35-52
Evaluation	**1. Reflective Questions**: How does using constants benefit a program? Why is it important to differentiate between constants and variables? **2. Practice Question:** Write a C program that demonstrates the use of constants and variables in a practical context.

Lesson Plan No. 2.5	Course Name: Programming in C Topic: Operators	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand and use various operators in C programming. b. Apply arithmetic, relational, logical, and bitwise operators. c. Differentiate between different types of operators and their use cases.
Teaching Aids (if any)	a. Slides with sample code and diagrams b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What types of operators have you used in other programming languages? - How do operators help in performing operations on data? **b. Introduction to Operators:** - Define operators: "Operators are symbols used to perform operations on variables and values." - Briefly discuss different types of operators. **Video Reference**: Operators in C Programming **2. Development (30 minutes)** **a. Arithmetic Operators (10 minutes):** - Overview: Explain operators like +, -, *, /, and %. - Examples: Show examples of arithmetic operations in C. **Video Reference:** Arithmetic Operators in C **b. Relational Operators (10 minutes):** - Overview: Discuss operators like ==, !=, <, >, <=, and >=. - Examples: Illustrate relational operations and their use in comparisons. **Video Reference**: Relational Operators in C **c. Logical and Bitwise Operators (10 minutes):**

	- Overview: Explain logical operators (&&, ||, !) and bitwise operators (&, |, ^, ~). - Examples: Show examples and practical uses of these operators. **Video Reference**: Logical and Bitwise Operators in C **3. Exercise (5 minutes)** - Activity: Have students write expressions using different operators. - Example Task: Create a program that performs arithmetic and relational operations and prints the results.
Closure	**1. Summarize key points:** - Importance of understanding and using operators in C programming. - Different types of operators and their applications. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 4: Operators, pp. 69-94 **Book 2:** "Programming with C" by Byron Gottfried Chapter 4: Operators and Expressions, pp. 53-72
Evaluation	1. **Reflective Questions:** Discuss why C is still relevant and how understanding its basics can aid in learning other programming languages. 2. **Practice Question:** Write a simple C program that demonstrates the use operators and basic I/O functions.

Lesson Plan No. 2.6	Course Name: Programming in C Topic: Precedence of Operators	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of operator precedence in C. b. Correctly evaluate expressions by applying the rules of operator precedence. c. Identify common pitfalls related to operator precedence.
Teaching Aids (if any)	a. Slides with operator precedence tables b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What happens when multiple operators are used in a single expression? - Why do you think operator precedence is important in programming? **b. Introduction to Operator Precedence:** - Define operator precedence: "Operator precedence determines the order in which operators are evaluated in an expression." - Discuss the importance of understanding precedence to avoid logical errors. **Video Reference**: Understanding Operator Precedence in C **2. Development (30 minutes)** **a. Precedence and Associativity (15 minutes):** - Overview: Explain the precedence and associativity of different operators. - Examples: Show examples of expressions with mixed operators and demonstrate the correct order of evaluation. **Video Reference**: Operator Precedence and Associativity **b. Common Pitfalls (10 minutes):** - Overview: Discuss common mistakes programmers make with operator precedence. - Examples: Illustrate incorrect and correct ways to evaluate expressions involving multiple operators.

	Video Reference: Common Mistakes with Operator Precedence **c. Practice with Examples (5 minutes):** - Activity: Solve example problems where students determine the correct order of operations. **Video Reference**: Practice Problems for Operator Precedence **3. Exercise (5 minutes)** - Activity: Have students evaluate complex expressions and write out the order of operations. - Example Task: Given an expression, identify the order in which operations will be performed and compute the result.
Closure	**1. Summarize key points:** - Importance of understanding and correctly applying operator precedence. - How precedence affects the outcome of expressions in C programming. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 5: Precedence and Associativity of Operators, pp. 95-115 **Book 2:** "Programming with C" by Byron Gottfried Chapter 5: Expressions and Precedence, pp. 73-89
Evaluation	**1. Reflective Questions:** Why is it crucial to understand operator precedence in programming? How can it prevent errors? **2. Practice Question:** Write and evaluate expressions that involve a combination of arithmetic, relational, and logical operators.

Lesson Plan No. 2.7	Course Name: Programming in C Topic: Statements and Expressions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Differentiate between statements and expressions in C programming. b. Write and evaluate different types of expressions. c. Understand the role of statements in structuring a C program.
Teaching Aids (if any)	a. Slides with sample code and diagrams b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What do you understand by the terms "statement" and "expression"? - How do you think expressions and statements contribute to a program? **b. Introduction to Statements and Expressions:** - Define statements and expressions: "An expression in C is a combination of variables, constants, and operators that results in a value. A statement is an instruction that the compiler executes." - Discuss the importance of understanding both concepts for effective programming. **Video Reference**: Understanding Expressions and Statements in C **2. Development (30 minutes)** **a. Expressions (15 minutes):** - Overview: Explain different types of expressions (arithmetic, relational, logical). - Examples: Show examples of expressions and their evaluation. **Video Reference**: Types of Expressions in C **b. Statements (10 minutes):** - Overview: Discuss different types of statements (declaration, assignment, control). - Examples: Illustrate how statements are used to structure a C

	program. **Video Reference**: Understanding Statements in C **c. Practical Applications (5 minutes):** - Overview: Discuss how expressions and statements work together in C programs. - Examples: Write simple programs demonstrating the use of various expressions and statements. **Video Reference**: Practical Use of Expressions and Statements **3. Exercise (5 minutes)** - **Activity:** Have students write a simple C program using different types of expressions and statements. - **Example Task**: Create a program that performs arithmetic operations and uses conditional statements to display results.
Closure	**1. Summarize key points:** - The role of expressions and statements in C programming. - How understanding these concepts is essential for writing effective and efficient code. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 6: Statements and Expressions, pp. 116-135 **Book 2:** "Programming with C" by Byron Gottfried Chapter 6: Control Statements and Expressions, pp. 90-112
Evaluation	**1. Reflective Questions:** How do expressions differ from statements? Why are both necessary for programming? **2. Practice Question**: Write a C program that includes different types of statements and expressions, and explain the flow of execution.

Lesson Plan No. 2.8	Course Name: Programming in C Topic: Output Functions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the purpose and usage of input-output functions in C. b. Utilize printf() and scanf() functions for basic input and output operations. c. Handle different data types with appropriate format specifiers in I/O functions.
Teaching Aids (if any)	a. Slides with examples of printf() and scanf() usage b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - How do you think a program can interact with the user? - Why is it important to display results and take input in programs? **b. Introduction to Input-Output Functions:** - Define input-output functions: "Input-output functions in C are used to take input from the user and display output on the screen." - Discuss the significance of printf() and scanf() as primary I/O functions in C. **Video Reference**: Introduction to Input-Output Functions in C **2. Development (30 minutes)** **a. The printf() Function (10 minutes):** - Overview: Explain the syntax and usage of printf() for displaying output. - Examples: Show how to print text, variables, and formatted data. **Video Reference**: How to Use printf() in C **b. The scanf() Function (10 minutes):** - Overview: Discuss the syntax and usage of scanf() for taking user input. - Examples: Demonstrate how to read different data types using

	format specifiers. **Video Reference**: Understanding scanf() in C **c. Handling Format Specifiers (10 minutes):** - Overview: Explain the role of format specifiers in both printf() and scanf(). - Examples: Provide examples of using format specifiers for integers, floats, characters, and strings. **Video Reference:** Working with Format Specifiers in C **3. Exercise (5 minutes)** - Activity: Have students write a program that takes user input and displays it using printf(). - Example Task: Create a program that reads an integer, a float, and a string from the user, and then prints them back in a formatted manner.
Closure	**1. Summarize key points:** - The role of printf() and scanf() in C programming. - Importance of format specifiers when handling different data types in I/O operations. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 4: Input-Output Functions, pp. 75-94 **Book 2:** "Programming with C" by Byron Gottfried Chapter 4: Input-Output in C, pp. 61-72
Evaluation	**1. Reflective Questions**: Why is it essential to understand input-output functions in C? How do they enhance user interaction with a program? **2. Practice Question**: Write a C program that uses printf() and scanf() to read and display different types of data.

Lesson Plan No. 3.1	Course Name: Programming in C Topic: Control Statements, Storage Classes, and Standard Library Function	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand and implement control statements for decision-making and branching. b. Apply looping constructs for repetitive tasks. c. Describe and use different storage classes and understand their scope and lifetime. d. Utilize standard library functions for input/output, string manipulation, character handling, mathematical operations, and time/date management.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment (e.g., an IDE like Code::Blocks, or an online compiler) d. Example code snippets e. Reference links for library functions
Teaching Development	**1. Introduction (10 minutes)** a. Pre-Discussion Questions: - What are control statements, and why are they important in programming? - Can you think of scenarios where different storage classes might be used? - How do standard library functions simplify programming tasks? b. Introduction to Control Statements, Storage Classes, and Library Functions: c. Brief overview of control statements: if-else, switch-case. d. Explanation of storage classes: automatic, static, external, and register. e. Introduction to standard library functions and their categories. **2. Development (30 minutes)** a. Control Statements (15 minutes): • Decision Making and Branching: - Explain if, if-else, else-if, and switch-case statements. - Provide syntax and examples. - Hands-on: Write simple programs to demonstrate each control statement. - Example: Program to check if a number is positive, negative, or zero. • Control Structures: Looping (10 minutes):

	- Explain for, while, and do-while loops. - Discuss use cases for each type of loop. - Hands-on: Write programs using loops (e.g., printing numbers 1 to 10). https://www.youtube.com/watch?v=6p7zS0XZJHc b. Storage Classes (10 minutes): • Types of Storage Classes: - Describe auto, static, extern, and register. - Discuss their scope, lifetime, and use cases. - Example: Code snippet demonstrating different storage classes. • Standard Library Functions (5 minutes): - Categories and Examples: - I/O Functions: printf, scanf - String Functions: strlen, strcpy, strcmp - Character Functions: isalnum, isalpha - Mathematics Functions: sqrt, pow, abs - Time and Date Functions: time, clock, strftime • Hands-on: Write a small program using standard library functions to perform basic tasks (e.g., calculate the factorial of a number, manipulate strings). c. Standard Library Functions (5 minutes): • Categories and Examples: - I/O Functions: printf, scanf - String Functions: strlen, strcpy, strcmp - Character Functions: isalnum, isalpha - Mathematics Functions: sqrt, pow, abs - Time and Date Functions: time, clock, strftime **3. Exercise (10 minutes)** Activity: - Control Statements: Students will complete a small set of problems involving decision-making and looping constructs. - Storage Classes: Implement a program that demonstrates the effects of different storage classes. - Standard Library Functions: Use library functions in a sample program (e.g., reading user input and performing operations on it).
Closure	1. Summarize Key Points: - Review the importance of control statements for flow control. - Recap storage classes and their impact on variable behavior. - Highlight how standard library functions can simplify

	coding tasks. 2. Suggested Readings from Textbooks: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy Chapter 2: "Control Statements" (pp. 11-25) Chapter 4: "Storage Classes" (pp. 45-60) **BOOK 2:** "Programming with C" by Byron Gottfried Chapter 3: "Functions" (pp. 30-50) Chapter 5: "Arrays and Strings" (pp. 75-90)
Evaluation	1. Reflective Questions: - How do control statements affect the flow of a program? - What are the implications of choosing different storage classes in a program? - How do standard library functions contribute to programming efficiency? 2. Homework: - Write a program that utilizes control statements, storage classes, and at least three different standard library functions. Document the code with comments explaining the purpose of each component.

Lesson Plan No. 3.2	Course Name: Programming in C Topic: Functions in C Programming	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and types of functions. b. Implement functions in C, including parameter passing and return values. c. Explore function prototypes and recursion.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment d. Example code snippets
Teaching Development	**1. Introduction (10 minutes)** a. Pre-Discussion Questions: - What is a function in programming, and why is it important? - How do functions help in breaking down complex tasks? b. Introduction to Functions: - Define what functions are and their role in C programming. YouTube Video: C Functions Tutoriall **2. Development (30 minutes)** a. Function Basics (10 minutes): - Syntax: function declaration, definition, and invocation. - Example: Simple function to add two numbers. b. Parameter Passing (10 minutes): - Pass-by-value vs. pass-by-reference. - Example: Demonstrate both types with code snippets. c. Recursion (10 minutes): - Define recursion and explain with examples (e.g., factorial calculation). **3. Exercise (10 minutes)** - Write a program with multiple functions demonstrating different parameter passing techniques. - Implement a recursive function for a common problem (e.g., Fibonacci series).
Closure	1. Recap the importance of functions, parameters, and recursion. 2. Suggested Readings: **BOOK 1**: “Programming in ANSI C” by E. Balagurusamy, Chapter 5: "Functions" **BOOK 2**: “Programming with C” by Byron Gottfried, Chapter 6: "Functions

Evaluation	1. Reflective Questions: What are the advantages of using functions in C? How does recursion differ from iteration? 2. Homework: Implement a set of functions to solve a specific problem (e.g., sorting an array).

Lesson Plan No.3.3	Course Name: Programming in C Topic: Arrays and Strings	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand and implement arrays and strings. b. Perform operations on arrays and strings. c. Demonstrate the use of multi-dimensional arrays.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment d. Example code snippets
Teaching Development	**1. Introduction (10 minutes)** a. Pre-Discussion Questions: - What is the purpose of arrays in programming? - How do you think strings are managed in C? b. Introduction to Arrays and Strings: - Define arrays and strings. **YouTube Video:**Arrays and Strings in C **2. Development (30 minutes)** a. Arrays (15 minutes): - Single-dimensional arrays: declaration, initialization, and operations. **YouTube Video**: Arrays in C Programming b. Strings (10 minutes): - String handling functions (strlen, strcpy, strcmp). **YouTube Video**: String Handling in C **c.** Multi-dimensional Arrays (5 **minutes):** - Declaration and initialization. **YouTube Video:**Multi-dimensional Arrays in C **3. Exercise (10 minutes)** - Write programs to perform various operations on arrays and strings. - Implement a program using multi-dimensional arrays for a practical application.
Closure	1. Summarize key points about arrays and strings. 2. Suggested Readings: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy, Chapter 6: "Arrays" **BOOK 2:** "Programming with C" by Byron Gottfried, Chapter 8: "Strings

Evaluation	1. Reflective Questions: How do arrays and strings facilitate data handling in C? What challenges are associated with multi-dimensional arrays? 2. Homework: Create a program that performs various operations on both arrays and strings.

Lesson Plan No. 3.4	Course Name: Introduction to C Programming Topic: Pointers and Dynamic Memory Allocation	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand pointers and their use in C programming. b. Implement dynamic memory allocation and deallocation. c. Explore common pointer-related issues and best practices.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment d. Example code snippets e. YouTube videos
Teaching Development	**1. Introduction (10 minutes)** a. Pre-Discussion Questions: - What do you understand by pointers in programming? - Why is dynamic memory allocation useful? b. Introduction to Pointers and Dynamic Memory Allocation: - Define pointers and their significance. **YouTube Video**: Pointers in C Programming **2. Development (30 minutes)** a. Pointers (15 minutes): - Pointer declaration, initialization, and dereferencing. **YouTube Video**: Understanding Pointers b. Dynamic Memory Allocation (15 minutes): - Functions: malloc, calloc, realloc, and free. **YouTube Video**: Dynamic Memory Allocation in C **3. Exercise (10 minutes)** - Implement a program that uses pointers for different tasks (e.g., pointer arithmetic). - Write a program demonstrating dynamic memory allocation for a data structure.
Closure	1. Recap the importance of pointers and dynamic memory management. 2. Suggested Readings: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy, Chapter 7: "Pointers" **BOOK 2**: "Programming with C" by Byron Gottfried, Chapter 10: "Pointers and Dynamic Memory Allocation"

Evaluation	1. Reflective Questions: How do pointers improve the efficiency of a program? What are the risks of improper memory management? 2. Homework: Develop a program that dynamically allocates and deallocates memory for different data structures.

Lesson Plan No. 3.5	Course Name: Programming in C Topic: File Handling	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of file handling in C. b. Implement file operations such as reading and writing. c. Explore different file modes and their uses.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment d. Example code snippets
Teaching Development	**1. Introduction (10 minutes)** a. Pre-Discussion Questions: - Why is file handling important in programming? - What operations might you perform on files? b. Introduction to File Handling: - Define file handling and its importance in C programming. **YouTube Video**: File Handling in C **2. Development (30 minutes)** a. File Operations (15 minutes): - Functions: fopen, fclose, fread, fwrite, fprintf, fscanf. **YouTube Video**: File Operations in C b. File Modes and Error Handling (15 minutes): - Explain file modes (e.g., "r", "w", "a"). - Discuss error handling in file operations. **YouTube Video**: Error Handling in File Operations **3. Exercise (10 minutes)** - Write programs to perform basic file operations (e.g., create, read, and write a text file). - Implement error handling for file operations.
Closure	1. Summarize key points about file handling. 2. Suggested Readings: **BOOK 1**: "Programming in ANSI C" by E. Balagurusamy, Chapter 8: "File Handling" **BOOK 2**: "Programming with C" by Byron Gottfried, Chapter 11: "File Handling".
Evaluation	1. Reflective Questions: What are the common challenges in file handling? How does file mode affect file operations? 2. Homework: Develop a program that handles various file

	operations, including error handling.

Lesson Plan No. 3.6	Course Name: Programming in C Topic: Pointers and Memory Management	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand pointers and their use in C. b. Learn about dynamic memory allocation. c. Write programs using pointers and dynamic memory functions.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What role do pointers play in managing memory in programs? - How do you think dynamic memory allocation impacts program efficiency? b. Introduction (15 minutes): - Overview: Explain pointers and dynamic memory management, highlighting their importance in managing and optimizing memory usage in C. **YouTube Video**: Pointers in C Programming (Pointers Tutorial) **2. Development (Brief):** a. Introduction to Pointers (20 minutes): - Explain pointers, pointer arithmetic, and dereferencing with examples. **YouTube Video**: Understanding Pointers in C b. Dynamic Memory Allocation (20 minutes): - Discuss malloc, calloc, realloc, and free with examples. **YouTube Video**: Dynamic Memory Allocation in C **3. Exercise (15 minutes):** - Write programs using pointers and dynamic memory functions.
Closure	1. Review pointers and memory management concepts. 2. Suggested readings: **Book 1**-"Programming in ANSI C" by E. Balagurusamy, Chapter 6, and **Book 2**-"Programming with C" by Byron Gottfried, Chapter 7
Evaluation	1. Reflective questions on pointers and memory management.

	2. 2. Homework: Develop a program that uses dynamic memory allocation to manage an array.

Lesson Plan No. 3.7	Course Name: Programming in C Topic: Structures and Unions	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand and use structures to manage complex data types. b. Utilize unions to handle different data types in the same memory location. c. Implement programs that use structures and unions effectively.
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction(5minutes)** **a.** Pre-Discussion Questions**:** - How can structures help in organizing complex data types? - What are the key differences between structures and unions? b. Introduction (15 minutes): - Overview: Introduce structures and unions, explaining how they help in managing complex data types and memory allocation. **YouTube Video**: Structures and Unions in C (Structures and Unions Tutorial) **2. Development (30 minutes)** a. Introduction to Structures (20 minutes): - Explain structure declaration, initialization, and member access with examples. **YouTube Video**: Structures in C Programming b. Introduction to Unions (20 minutes): - Discuss union declaration, usage, and advantages over structures. **YouTube Video**: Unions in C **3. Exercise (15 minutes):** - Implement programs that use structures and unions to manage data.
Closure	1. Review structures and unions. 2. Suggest readings: **Book 1**-"Programming in ANSI C" by E. Balagurusamy, Chapter 7, and **Book 2**-"Programming with C" by Byron Gottfried, Chapter 8.
Evaluation	1. Reflective questions on structures and unions.

	2. Homework: Write a program that uses structures and unions to manage and manipulate data

Lesson Plan No. 3.8	Course Name: Programming in C Topic: Advanced File Handling	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Explore advanced file handling techniques, including file pointers and binary files. b. Implement file manipulation functions to manage file pointers. c. Write programs that utilize advanced file handling concepts
Teaching Aids (if any)	a. Slides with diagrams and examples b. Use of Nearpod tool for online quiz
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: - What advanced file operations might be useful in more complex applications? - How can handling binary files differ from handling text files? b. Introduction (15 minutes): - Overview: Discuss advanced file handling techniques such as file pointers, binary file operations, and file manipulation functions. **YouTube Video**: Advanced File Handling in C (Advanced File Handling Tutorial) **2. Development (30 minutes):** a. Binary Files (20 minutes): - Explain binary file operations (fwrite, fread) and their use cases. **YouTube Video**: Binary File Handling in C b. File Manipulation Functions (20 minutes): - Discuss functions like fseek, ftell, and rewind for manipulating file pointers. **YouTube Video**: File Manipulation Functions **3. Exercise (15 minutes):** - Write programs using advanced file operations and binary files.
Closure	1. Review the advanced file handling concepts covered, including binary file operations and file pointer functions. 2. Suggest readings: **Book 1**-"Programming in ANSI C" by E. Balagurusamy, Chapter 8 **Book 2**-"Programming with C" by Byron Gottfried, Chapter 11

Evaluation	**1.** Reflective Questions: - What are the advantages of using binary files over text files? - How do file pointer functions improve file manipulation? **2.** Homework: Develop a program that involves complex file operations, such as reading from and writing to binary files, and includes comprehensive error handling. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 3.9	Course Name: Programming in C Topic: Storage Classes in C	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Understand the different types of storage classes in C and their purposes b. Explain the scope and lifetime of variables for each storage class c. Apply storage classes in practical scenarios to manage variables effectively.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: - What role do storage classes play in managing variables in a C program? - How does the scope and lifetime of a variable affect its usage? b. Introduction (10 minutes): - Overview: Introduce the concept of storage classes and their significance in variable management. Explain that different storage classes control the scope (visibility) and lifetime (duration) of variables. **2. Development (35 minutes):** a. Types of Storage Classes (20 minutes): Auto (Automatic): - Concepts: Default storage class for local variables; scope is limited to the block in which it is declared; lifetime is the duration of the block's execution. - Example: Show a simple example of an auto variable and explain its behaviour. Register: - Concepts: Suggests that a variable be stored in a CPU register for faster access; scope is limited to the block in which it is declared; lifetime is the duration of the block's execution. - Example: Provide a code example using register and discuss its potential performance benefits. Static: - Concepts: Retains its value between function calls; scope is limited to the block in which it is declared, but lifetime extends for the entire duration of the program. - Example: Demonstrate how a static variable maintains its

	value between function calls. Extern: - Concepts: Used to declare global variables that are accessible across multiple files; scope is global; lifetime is the entire eduration of the program. - Example: Show an example of using extern to access a global variable defined in another file. - Interactive Coding: Guide students through coding examples for each storage class, highlighting differences in scope and lifetime. b. Scope and Lifetime (10 minutes): Concepts: - Scope: Refers to the region of code where a variable can be accessed. - Lifetime: Refers to the duration a variable exists in memory. - Examples: Use visual aids and code examples to illustrate how scope and lifetime affect variable access and persistence. - Diagram: Provide diagrams showing the scope and lifetime of variables for different storage classes. https://www.youtube.com/watch?v=6Fcs3H5zXfE c. Practical Application (5 minutes): - Concepts: Discuss scenarios where each storage class might be used effectively. - Examples: Provide real-world examples, such as maintaining state information in functions or sharing global configuration settings across files. **3. Exercise (15 minutes):** a. Task 1: Code Exercise on Storage Classes: - Write a program that demonstrates the use of auto, register, static, and extern variables. Include different functions and files to show how each storage class affects variable behavior. b. Task 2: Scope and Lifetime Quiz: - Create a quiz with questions about the scope and lifetime of variables. Include scenarios where students need to identify which storage class is appropriate based on variable usage.
Closure	Review: 1. Summarize the key concepts: the types of storage classes, their scope and lifetime, and practical usage. Suggested Readings: **Book 1**-"Programming in ANSI C" by E. Balagurusamy, Chapter 6 **Book 2**-"Programming with C" by Byron Gottfried, Chapter 7

Evaluation	**1.** Reflective Questions: - Ask students to explain how the scope of a variable affects its visibility and accessibility in a program. - Discuss the lifetime of variables and how it influences their persistence across function calls. **2.** Homework: - Project Assignment: Develop a program that uses all four storage classes. The program should include functions that demonstrate the persistence of static variables, the visibility of extern variables, and the potential performance benefits of register variables. Provide documentation or comments in the code explaining the use of each storage class. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No.3.10	Course Name: Programming in C Topic: Standard Library Functions in C	Course No.: BCAMJ-101

Objectives	At the end of the lesson the student shall be able to: a. Utilize standard library functions for input/output operations. b. Apply string manipulation functions to handle and process strings effectively. c. Use character handling functions to perform operations on individual characters. d. Implement mathematical functions for calculations and numeric operations. e. Work with time and date functions for handling and formatting time-related data.
Teaching Aids (if any)	a. Whiteboard and markers b. Presentation slides c. Coding environment
Teaching Development	**1. Introduction (5 minutes):** a. Pre-Discussion Questions: - What are standard library functions and why are they important in programming? - How can using library functions improve the efficiency and readability of your code? b. Introduction (10 minutes): - Overview: Introduce the concept of standard library functions and their role in simplifying common programming tasks. Explain the benefits of using these pre-defined functions for I/O operations, string manipulation, character handling, mathematics, and date/time operations. - **2. Development (35 minutes):** a. I/O Functions (10 minutes): • Concepts: - Explain functions such as printf(), scanf(), fprintf(), and fscanf() for formatted input and output. • Examples: - Demonstrate basic usage with examples such as printing formatted text and reading user input. - Interactive Coding: Guide students through writing simple programs using I/O functions for user interaction. b. String Functions (10 minutes): • Concepts: - Discuss functions like strlen(), strcpy(), strcmp(), and strcat() for handling strings.

- Examples:
  - Provide examples of string manipulation, including copying strings, comparing strings, and concatenating strings.

- Interactive Coding: Have students write programs that manipulate strings, such as reversing a string or concatenating multiple strings.

c. Character Functions (5 minutes):

- Concepts:
  - Explain functions like isalpha(), isdigit(), toupper(), and tolower() for character handling.
- Examples:
  - Show examples of checking character types and converting characters to uppercase or lowercase.

YouTube Video: "Character Handling Functions in C Programming"

- Interactive Coding: Guide students through examples that involve character checks and conversions.

d. Mathematical Functions (5 minutes):

- Concepts:
  - Discuss functions such as abs(), sqrt(), pow(), sin(), and cos() for performing mathematical operations.
- Examples:
  - Demonstrate calculations using mathematical functions, such as computing square roots or powers.

YouTube Video: "Mathematical Functions in C Programming"

- Interactive Coding: Have students implement calculations using mathematical functions in their programs.

e. Time and Date Functions (5 minutes):

- Concepts:
  - Explain functions like time(), localtime(), and strftime() for handling time and date.
- Examples:
  - Show how to get the current time, format it, and display it in a readable format.

YouTube Video: "Time and date https://www.youtube.com/watch?v=4x2YRxA0aCk

- Date Functions in C Programming"
- Interactive Coding: Guide students through creating a program that displays the current date and time.

3. Exercise (15 minutes):

- Task 1: I/O Functions Exercise:
  - Write a program that uses printf() and scanf() to perform formatted input and output, such as creating a simple calculator.
- Task 2: String Functions Exercise:

	- Write a program to manipulate strings, such as reversing a string or finding the length of a string. • Task 3: Mathematical Functions Exercise: - Write a program that performs various mathematical calculations, such as computing the square root or power of a number. • Task 4: Time and Date Functions Exercise: - Write a program that displays the current date and time and formats it in a user-friendly way.
Closure	1. Review: - Summarize the key concepts: I/O functions, string manipulation, character handling, mathematical functions, and time/date functions. 2. Suggested Readings: **Book 1**- "Programming in ANSI C" by E. Balagurusamy, Chapters 7, 8, 9, and 10 **Book 2**-"Programming with C" by Byron Gottfried, Chapters 7, 8, 9, and 10
Evaluation	1. Reflective Questions: - Ask students to explain the importance of using standard library functions and how they can simplify programming tasks. - Discuss specific scenarios where different types of library functions might be used. 2. Homework: - Project Assignment: Develop a comprehensive program that integrates various standard library functions. The program should include input/output operations, string manipulations, character handling, mathematical calculations, and time/date functions. Provide comments in the code explaining the use of each library function.

Lesson Plan No. 4.1	Course Name: Programming in C Topic: User-Defined Functions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of user-defined functions in C. b. Define and implement functions. c. Differentiate between user-defined and built-in functions.
Teaching Aids (if any)	a. Slides with examples of function definitions and calls b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What are functions in general? - Why might we need to define our own functions in a program? **b.** Introduction to User-Defined Functions: - Define user-defined functions: "Functions created by the programmer to perform specific tasks in a program." **2. Development (30 minutes)** **a. Defining Functions (10 minutes):** - Overview: Explain the syntax for defining a function. - Examples: Show how to create simple functions like calculating the square of a number. **b. Function Calls (10 minutes):** - Overview: Explain how to call a user-defined function within the main program. - Examples: Demonstrate function calls with different arguments. **c. Return Values (10 minutes):** - Overview: Explain the concept of return values in functions. - Examples: Show how to return results from functions. **3. Exercise (5 minutes)** - Activity: Have students write a function to calculate the

	factorial of a number.
Closure	**1. Summarize key points:** - The significance of user-defined functions in modular programming. - Basic syntax and structure of defining and calling functions in C. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 5: Functions, pp. 95-124 **Book 2:** "Programming with C" by Byron Gottfried Chapter 5: User-Defined Functions, pp. 73-102
Evaluation	1. **Reflective Questions**: Why is it beneficial to create user-defined functions? 2. **Practice Question**: Write a program that uses a user-defined function to sum two numbers.

Lesson Plan No. 4.2	Course Name: Programming in C Topic: Arrays	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand and use various operators in C programming. b. Apply arithmetic, relational, logical, and bitwise operators. c. Differentiate between different types of operators and their use cases.
Teaching Aids (if any)	a. Slides with operator types and examples b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What types of operators have you used in other programming languages? - How do operators help in performing operations on data? **b. Introduction to Operators:** - Define operators: "Operators are symbols used to perform operations on variables and values." - Briefly discuss different types of operators. **Video Reference**: Operators in C Programming **2. Development (30 minutes)** **a. Arithmetic Operators (10 minutes):** - Overview: Explain operators like +, -, *, /, and %. - Examples: Show examples of arithmetic operations in C. **Video Reference**: Arithmetic Operators in C **b. Relational Operators (10 minutes):** - Overview: Discuss operators like ==, !=, <, >, <=, and >=. - Examples: Illustrate relational operations and their use in comparisons. **Video Reference:** Relational Operators in C **c. Logical and Bitwise Operators (10 minutes):**

	- Overview: Explain logical operators (&&, ||, !) and bitwise operators (&, |, ^, ~). - Examples: Show examples and practical uses of these operators. **Video Reference**: Logical and Bitwise Operators in C **3. Exercise (5 minutes)** - **Activity**: Have students write expressions using different operators. - **Example Task**: Create a program that performs arithmetic and relational operations and prints the results
Closure	**1. Summarize key points:** - The role of arrays in storing and managing data. - Basic operations such as declaration, initialization, and element access. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 7: Arrays, pp. 139-160 **Book 2:** "Programming with C" by Byron Gottfried Chapter 6: Arrays, pp. 103-128
Evaluation	**1. Reflective Questions:** How do arrays help in managing large amounts of data? **2. Practice Question**: Write a program to calculate the average of elements in an array.

Lesson Plan No. 4.3	Course Name: Programming in C Topic: Recursion	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of recursion in programming. b. Implement recursive functions. c. Analyze the advantages and limitations of using recursion
Teaching Aids (if any)	a. Slides with examples of recursive functions b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What do you understand by the term "recursion"? - Can you think of a problem that can be solved by breaking it down into smaller similar problems? **b. Introduction to Recursion:** - **Define recursion**: "Recursion is a programming technique where a function calls itself to solve a smaller instance of the problem." **2. Development (30 minutes)** **a. Recursive Function Basics (10 minutes):** - Overview: Explain the basic structure of a recursive function. - Examples: Show examples like factorial calculation using recursion. **b. Base Case and Recursive Case (10 minutes):** - Overview: Discuss the importance of having a base case to avoid infinite recursion. - Examples: Demonstrate how to set up base and recursive cases in different scenarios. **c. Recursive Function Design (10 minutes):** - Overview: Explain how to design a recursive function by breaking down problems into smaller sub-problems. - Examples: Show how to solve problems like the Fibonacci sequence using recursion.

	3. Exercise (5 minutes) - **Activity**: Have students write a recursive function to calculate the greatest common divisor (GCD) of two numbers.
Closure	**1. Summarize key points:** - The concept and utility of recursion in programming. - Designing effective recursive functions with base and recursive cases. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 5: Recursion, pp. 125-134 **Book 2:** "Programming with C" by Byron Gottfried Chapter 9: Recursion, pp. 150-168
Evaluation	**1. Reflective Questions**: What are the advantages and drawbacks of using recursion? **2. Practice Question**: Write a recursive program to reverse a string.

Lesson Plan No. 4.4	Course Name: Programming in C Topic: Handling of Character Strings	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of character strings in C. b. Declare and initialize string variables. c. Perform basic string operations like concatenation, comparison, and length determination.
Teaching Aids (if any)	a. Slides with examples of string operations b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is a string in programming? - How do you think strings are handled differently from numbers? **b. Introduction to Character Strings:** - Define character strings: "Strings are sequences of characters stored as an array in C." **2. Development (30 minutes)** **a. String Declaration and Initialization (10 minutes):** - Overview: Explain how to declare and initialize string variables in C. - Examples: Show examples of different ways to initialize strings. **b. String Operations (10 minutes):** - Overview: Explain common string operations such as concatenation, comparison, and finding the length. - Examples: Demonstrate string operations using built-in functions like strcat, strcmp, and strlen. **c. String Input/Output (10 minutes):** - Overview: Explain how to read and display strings using gets() and puts(). - Examples: Show examples of basic string input and output operations. **3. Exercise (5 minutes)** • Activity: Have students write a program to compare two strings

	entered by the user.
Closure	**1. Summarize key points:** - The importance of strings in handling textual data. - Basic string operations and their implementation in C. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 8: Strings, pp. 161-180 **Book 2:** "Programming with C" by Byron Gottfried Chapter 7: Handling Strings, pp. 129-148
Evaluation	**1. Reflective Questions**: Why are strings important in programming, and how do they differ from other data types? **2. Practice Question**: Write a program to reverse a string entered by the user.

Lesson Plan No. 4.5	Course Name: Programming in C Topic: Structures	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of structures in C. b. Declare and initialize structure variables. c. Access and manipulate data within structures.
Teaching Aids (if any)	a. Slides with examples of structure declarations and operations b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What do you understand by data structures? - How do structures differ from arrays? **b. Introduction to Structures:** - **Define structures**: "A structure is a user-defined data type in C that allows grouping of variables of different data types under a single name." **Video Reference:** Understanding Structures in C Programming **2. Development (30 minutes)** **a. Declaring and Initializing Structures (10 minutes):** - Overview: Explain the syntax for declaring structures and how to initialize them. - Examples: Show how to create a structure to store information like student details (name, age, grade). **Video Reference:** Declaring and Accessing Structures in C **b. Accessing Structure Members (10 minutes):** - Overview: Explain how to access and modify data within a structure using the dot operator. - Examples: Demonstrate accessing and manipulating structure members. **c. Nested Structures (10 minutes):** - Overview: Discuss the concept of nested structures, where a

	structure contains another structure. - Examples: Show examples of nested structures, such as a structure for a date (day, month, year) within a student record. **Video Reference:** Working with Arrays of Structures in C **3. Exercise (5 minutes)** - **Activity:** Have students write a program to create a structure for storing information about a book (title, author, price).
Closure	**1. Summarize key points:** - The role of structures in organizing complex data. - Basic operations on structure members and the concept of nested structures. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 9: Structures, pp. 181-204 **Book 2:** "Programming with C" by Byron Gottfried Chapter 8: Structures, pp. 149-170 **YouTube Links** • Introduction to Structures in C • Nested Structures in C
Evaluation	**1. Reflective Questions**: How do structures help in managing related data of different types? **2. Practice Question**: Write a program to create and display information about an employee using structures.

Lesson Plan No. 4.6	Course Name: Programming in C Topic: Unions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of unions in C. b. Declare and use union variables. c. Differentiate between structures and unions.
Teaching Aids (if any)	a. Slides with examples of union declarations and operations b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is a union in programming? - How do you think a union might differ from a structure? **b. Introduction to Unions:** - Define unions: "A union is a user-defined data type in C that allows storing different data types in the same memory location." **Video Reference:** Unions in C Programming **2. Development (30 minutes)** **a. Declaring and Using Unions (10 minutes):** - Overview: Explain the syntax for declaring unions and how to use them. - Examples: Show how to create a union for storing different data types (int, float, char) in a single memory location. **Video Reference:** Declaring and Accessing Unions in C **b. Accessing Union Members (10 minutes):** - Overview: Explain how to access data within a union. - Examples: Demonstrate accessing and manipulating union members. **c. Difference between Structures and Unions (10 minutes):** - Overview: Discuss the key differences between structures and unions in terms of memory usage and member access.

	- Examples: Compare and contrast with examples. **Video Reference:** Difference Between Structure and Union in C **3. Exercise (5 minutes)** - Activity: Have students write a program to create a union for storing different data types and display the stored data
Closure	**1. Summarize key points:** - The concept of unions and their efficient use of memory. - Differences between structures and unions. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy Chapter 10: Unions, pp. 205-218 **Book 2:** "Programming with C" by Byron Gottfried Chapter 9: Unions, pp. 171-185
Evaluation	**1. Reflective Questions**: Why might you choose to use a union instead of a structure? **2. Practice Question**: Write a program to demonstrate the use of unions to store and display different types of data.

Lesson Plan No. 4.7	Course Name: Programming in C Topic: Pointers to Structures and Unions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand how to use pointers with structures and unions. b. Access and manipulate structure and union members using pointers. c. Implement complex data structures using pointers to structures.
Teaching Aids (if any)	a. Slides with examples of pointers to structures and unions b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - How do pointers enhance the functionality of structures and unions? - Why might a programmer use a pointer to a structure instead of the structure itself? **b.** Introduction to Pointers to Structures and Unions: - Define pointers to structures and unions: "A pointer to a structure or union allows indirect access to the members of the structure or union using the pointer." - Discuss the significance of using pointers with complex data structures. **Video Reference:** Pointers to Structures in C Programming **2. Development (30 minutes)** **a. Declaring and Using Pointers to Structures (10 minutes):** - Overview: Explain how to declare a pointer to a structure and access its members using the arrow (->) operator. - Examples: Show how to define a pointer to a student structure and access its fields. **Video Reference:** Using Pointers with Structures in C **b. Working with Pointers to Unions (10 minutes):** - Overview: Discuss how to declare a pointer to a union and access its members. - Examples: Demonstrate how to manage memory and access union members using pointers.

	Video Reference: Pointers and Unions in C **c. Complex Data Structures with Pointers (10 minutes):** - Overview: Explain how pointers to structures can be used to create linked lists, trees, and other complex data structures. - Examples: Illustrate a simple linked list implementation using pointers to structures. **Video Reference:** Implementing Linked Lists in C Using Pointers **3. Exercise (5 minutes)** - **Activity**: Have students write a C program that defines a structure and accesses its members using a pointer. - **Example Task**: Create a program that dynamically allocates memory for a structure and prints its members using a pointer.
Closure	**1. Summarize key points:** - The concept and importance of using pointers with structures and unions in C programming. - Practical applications of pointers in complex data structures. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy, Chapter 9: Pointers to Structures and Unions, pp. 194-210. **Book 2:** "Programming with C" by Byron Gottfried, Chapter 10: Pointers and Data Structures, pp. 226-240.
Evaluation	**1. Reflective Questions:** How do pointers enhance the power and flexibility of structures and unions in C programming? What are the potential pitfalls of using pointers? **2. Practice Question:** Write a C program that uses a pointer to a structure to implement a simple student database and print the details of the students.

Lesson Plan No. 4.8	Course Name: Programming in C Topic: User-Defined Functions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of user-defined functions in C. b. Define and call user-defined functions. c. Pass arguments to functions and return values from functions.
Teaching Aids (if any)	a. Slides with examples of pointers to structures and unions b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is a function in programming? - How can functions make a program more modular and manageable? **b. Introduction to User-Defined Functions:** - Define user-defined functions: "User-defined functions in C are functions created by the programmer to perform specific tasks." - Discuss the importance of using functions to break down complex problems into simpler tasks. **Video Reference:** Introduction to Functions in C **2. Development (30 minutes)** **a. Defining and Calling Functions (10 minutes):** - **Overview**: Explain how to declare a function, define its body, and call it from main(). - **Examples:** Show how to create a simple function to calculate the sum of two integers. **Video Reference:** How to Define and Call Functions in C **b. Passing Arguments to Functions (10 minutes):** - **Overview**: Discuss how to pass arguments to a function and use them within the function body. - **Examples**: Demonstrate passing different types of arguments (int, float, etc.) to a function. **Video Reference:** Passing Arguments to Functions in C

	c. Returning Values from Functions (10 minutes): - Overview: Explain how a function can return a value to the calling code. - Examples: Show how to return the result of a calculation from a function. **Video Reference:** Returning Values from Functions in C **3. Exercise (5 minutes)** - **Activity**: Have students write a C program that defines a function to calculate the area of a rectangle and returns the result. - **Example Task**: Create a function int area(int length, int width) that takes two integers as parameters and returns the area.
Closure	**1. Summarize key points:** - The concept and importance of user-defined functions in C programming. - How to define, call, and return values from functions. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy, Chapter 5: Functions, pp. 101-130. **Book 2:** "Programming with C" by Byron Gottfried, Chapter 5: Functions in C, pp. 123-145.
Evaluation	**1. Reflective Questions:** How do user-defined functions improve the modularity and readability of a program? What are the benefits of returning values from a function? **2. Practice Question:** Write a C program that defines a function to calculate the factorial of a number using recursion and returns the result.

Lesson Plan No. 4.9	Course Name: Programming in C Topic: Arrays	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of arrays in C. b. Declare, initialize, and access elements of one-dimensional arrays. c. Utilize arrays to store and manipulate collections of data.
Teaching Aids (if any)	a. Slides with examples of array declarations and operations b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is an array in programming? - Why might you need to store multiple values of the same type? **b. Introduction to Arrays:** - Define arrays: "An array in C is a collection of elements of the same type stored in contiguous memory locations." - Discuss the advantages of using arrays for managing multiple related variables. **Video Reference:** Introduction to Arrays in C **2. Development (30 minutes)** **a. Declaring and Initializing Arrays (10 minutes):** - Overview: Explain how to declare an array and initialize it with values. - Examples: Show how to declare an integer array and initialize it with specific values. **Video Reference:** Declaring and Initializing Arrays in C **b. Accessing and Modifying Array Elements (10 minutes):** - Overview: Discuss how to access and modify individual elements of an array using indices. - Examples: Demonstrate how to change the value of an array element at a specific index. **Video Reference:** Accessing Array Elements in C **c. Using Arrays in Programs (10 minutes):**

	- Overview: Explain how arrays can be used in loops and other constructs to manage multiple values. - Examples: Illustrate a simple program that calculates the average of an array of integers. **Video Reference:** Using Arrays in C Programming **3. Exercise (5 minutes)** - Activity: Have students write a C program that declares an array of integers, fills it with values, and prints the sum of the elements. - Example Task: Create a program that declares an array of 5 integers, assigns values to each element, and prints the total sum.
Closure	**1. Summarize key points:** - The concept and importance of arrays in C programming. - How to declare, initialize, and access array elements. **2. Suggested Readings** **Book 1**: "Programming in ANSI C" by E. Balagurusamy, Chapter 6: Arrays, pp. 131-150. **Book 2**: "Programming with C" by Byron Gottfried, Chapter 6: Arrays in C, pp. 146-168.
Evaluation	**1. Reflective Questions:** How do arrays help in managing multiple variables of the same type? What are the potential pitfalls of using arrays? **2. Practice Question:** Write a C program that finds the maximum and minimum values in an array of integers.

Lesson Plan No. 4.10	Course Name: Programming in C Topic: Recursion	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of recursion in C programming. b. Implement recursive functions. c. Recognize the difference between recursion and iteration.
Teaching Aids (if any)	a. Slides with examples of recursive functions b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is recursion in mathematics or daily life? - Can you think of a problem that can be solved by breaking it down into smaller, similar problems? **b. Introduction to Recursion:** - Define recursion: "Recursion in C is a process where a function calls itself directly or indirectly to solve a problem." - Discuss the importance of recursion for solving problems that can be broken down into simpler, similar sub-problems. **Video Reference:** Introduction to Recursion in C **2. Development (30 minutes)** **a. Writing Recursive Functions (10 minutes):** - Overview: Explain how to write a recursive function with a base case and recursive step. - Examples: Show how to write a function to calculate the factorial of a number using recursion. **Video Reference:** Writing Recursive Functions in C **b. Recursion vs. Iteration (10 minutes):** - Overview: Discuss the differences between recursion and iteration, including when to use each approach. - Examples: Demonstrate the difference by comparing a recursive and iterative solution for calculating the Fibonacci sequence. **Video Reference:** Recursion vs Iteration in C **c. Applications of Recursion (10 minutes):**

	- Overview: Explain where recursion is particularly useful, such as in tree traversals, solving puzzles, and searching algorithms. - Examples: Illustrate how recursion is used in algorithms like QuickSort and MergeSort. **Video Reference:** Applications of Recursion in C **3. Exercise (5 minutes)** - Activity: Have students write a recursive C program that calculates the nth Fibonacci number. - Example Task: Create a function int fibonacci(int n) that returns the nth Fibonacci number using recursion.
Closure	**1. Summarize key points:** - The concept and importance of recursion in C programming. - How to write recursive functions and understand their base and recursive cases. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy, Chapter 7: Recursion, pp. 151-170. **Book 2**: "Programming with C" by Byron Gottfried, Chapter 7: Recursion in C, pp. 169-188.
Evaluation	**1. Reflective Questions:** What are the advantages and disadvantages of using recursion over iteration? In what scenarios is recursion more suitable? **2. Practice Question:** Write a C program that uses recursion to solve the Tower of Hanoi problem.

Lesson Plan No. 4.11	Course Name: Programming in C Topic: Handling of Character Strings	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of character strings in C. b. Declare, initialize, and manipulate character strings. c. Use string handling functions to perform operations on strings.
Teaching Aids (if any)	a. Slides with examples of string declarations and operations b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is a string in everyday language? How does it differ in programming? - Why is handling text data important in most applications? **b. Introduction to Character Strings:** - Define character strings: "A string in C is an array of characters terminated by a null character (\0)." - Discuss the importance of strings in handling textual data in programs. **Video Reference:** Introduction to Strings in C **2. Development (30 minutes)** **a. Declaring and Initializing Strings (10 minutes):** - Overview: Explain how to declare a string and initialize it with text. - Examples: Show how to declare a character array and initialize it with a string literal. **Video Reference:** Declaring and Initializing Strings in C **b. String Manipulation Functions (10 minutes):** • Overview: Discuss standard string handling functions like strlen, strcpy, strcat, and strcmp. • Examples: Demonstrate how to use these functions to perform basic string operations. **Video Reference:** String Manipulation in C

	c. Common String Operations (10 minutes): - Overview: Explain common operations like string concatenation, comparison, and searching within a string. - Examples: Illustrate how to concatenate two strings and compare them using standard functions. **Video Reference:** Common String Operations in C **3. Exercise (5 minutes)** - Activity: Have students write a C program that reads a string from the user, reverses it, and prints the reversed string. - Example Task: Create a program that reads a string and prints it in reverse order using a loop.
Closure	**1. Summarize key points:** - The concept and importance of character strings in C programming. - How to declare, initialize, and manipulate strings using standard functions. **2. Suggested Readings** **Book 1:** "Programming in ANSI C" by E. Balagurusamy, Chapter 8: Strings, pp. 171-193. **Book 2**: "Programming with C" by Byron Gottfried, Chapter 8: String Handling in C, pp. 189-211.
Evaluation	**1. Reflective Questions:** Why are strings important in programming? How can you efficiently manipulate strings in C? **2. Practice Question:** Write a C program that counts the number of vowels and consonants in a given string.

Lesson Plan No. 4.12	Course Name: Programming in C Topic: Multidimensional Array Declaration and Their Applications	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept and syntax of multidimensional arrays in C. b. Declare and initialize multidimensional arrays. c. Apply multidimensional arrays in practical scenarios like matrix operations.
Teaching Aids (if any)	a. Slides with examples of multidimensional array declarations b. Chalkboard/Whiteboard
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - Have you worked with single-dimensional arrays before? - How would you represent a table or matrix in programming? **b. Introduction to Multidimensional Arrays:** - Define multidimensional arrays: "Multidimensional arrays in C are arrays of arrays, allowing storage of data in a grid-like structure." - Discuss the significance of using multidimensional arrays for tasks like matrix operations and data representation in tables. **Video Reference:** Introduction to Multidimensional Arrays in C **2. Development (30 minutes)** **a. Declaring and Initializing Multidimensional Arrays (10 minutes):** - Overview: Explain the syntax for declaring and initializing a 2D array. - Examples: Show how to create a 3x3 matrix and initialize it with values. **Video Reference:** How to Declare and Initialize 2D Arrays in C **b. Accessing and Modifying Elements in a Multidimensional Array (10 minutes):** - Overview: Demonstrate how to access and modify elements in a 2D array using row and column indices. - Examples: Show examples of how to update values in a matrix and print the matrix elements.

	Video Reference: Accessing and Modifying 2D Array Elements in C **c. Applications of Multidimensional Arrays (10 minutes):** - Overview: Discuss practical applications like matrix multiplication, image processing, and storing tabular data. - Examples: Implement a simple matrix addition program using a 2D array. **Video Reference:** Matrix Operations Using Multidimensional Arrays **3. Exercise (5 minutes)** - Activity: Have students write a program that creates a 3x3 matrix, fills it with user input, and then prints the matrix. - Example Task: Create a program that performs matrix addition using two 2D arrays.
Closure	**1. Summarize key points:** - The role of multidimensional arrays in C programming. - How to declare, initialize, and access elements in a multidimensional array. - Practical applications of multidimensional arrays in matrix operations and data representation. **2. Suggested Readings** **Book 1**: "Programming in ANSI C" by E. Balagurusamy Chapter 6: Arrays, pp. 120-145 **Book 2**: "Programming with C" by Byron Gottfried Chapter 7: Arrays, pp. 158-180
Evaluation	**1. Reflective Questions** - Why are multidimensional arrays essential for matrix operations? - How does understanding multidimensional arrays help in solving complex data representation problems? **2. Practice Question:** - Write a C program that uses a 3x3 matrix to perform matrix addition and prints the result.

Lesson Plan No. 5.1	Course Name: Introduction to C programming Topic: Understanding Pointers in C Programming	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Define and explain the concept of pointers in C. b. Understand the syntax and structure of pointer variables. c. Differentiate between normal variables and pointer variables. d. Apply pointers in simple C programs.
Teaching Aids (if any)	a. Slides with diagrams illustrating pointer concepts and memory addresses. b. Chalkboard/Whiteboard for explaining pointer notation.

Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What do you understand by the term "pointer"? - How do you think a pointer might be useful in programming? - What challenges might arise when using pointers in C? **b. Introduction to Pointers:** - Define a pointer: "A pointer is a variable that stores the memory address of another variable." - Discuss the importance of pointers in efficient memory management and dynamic allocation. ***Video Reference***: Introduction to Pointers in C **2. Development (30 minutes)** **a. Pointer Declaration and Initialization (10 minutes)** - **Overview**: Explain how to declare and initialize a pointer, and how it differs from a normal variable. - **Examples**: Demonstrate using a pointer to store the address of an int variable. ***Video Reference***: Pointer Declaration and Initialization **b. Pointer Dereferencing (10 minutes)** - **Definition:** Explain the concept of dereferencing and how it is used to access the value stored in the memory address a pointer

	refers to. - **Examples:** Show examples of dereferencing to access and modify the values of variables through pointers. **Video Reference**: Understanding Pointer Dereferencing **c. Pointer Arithmetic (10 minutes)** - **Overview:** Discuss pointer arithmetic, including adding and subtracting integers to/from pointers to traverse arrays. - **Examples**: Show how pointer arithmetic can be applied to iterate through arrays. **Video Reference**: Pointer Arithmetic in C **3. Exercise (5 minutes)** - **Activity**: Have students write a C program that declares a pointer, stores the address of an integer variable, and prints the dereferenced value. - **Example Task**: Create a program that declares an int variable, assigns a value, and uses a pointer to print and modify this value.
Closure	**1. Summarize key points:** - Define what a pointer is and why it is crucial for memory management in C programming. - Recap the differences between normal variables and pointers. - Highlight the concept of pointer arithmetic and its use in arrays. **2. Suggested Readings** - **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 6: Pointers, pp. 140-170 - **Book 2: "Programming with C" by Byron Gottfried** Chapter 7: Pointers and Arrays, pp. 235-264 - **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 5: Pointers and Arrays, pp. 90-120
Evaluation	**1. Reflective Questions:** - How do pointers improve the efficiency of memory usage? - What are some common challenges when working with pointers in C? **2. Practice Question:** - Write a C program that demonstrates pointer arithmetic by creating an array, using pointers to traverse it, and printing the values.

Lesson Plan No. 5.2	Course Name: Programming in C Topic: Dynamic Memory Allocation in C	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of dynamic memory allocation in C. b. Use functions like malloc(), calloc(), realloc(), and free() effectively. c. Differentiate between static and dynamic memory allocation. d. Apply dynamic memory allocation in practical C programs.
Teaching Aids (if any)	a. Slides explaining memory allocation concepts. b. Video clips demonstrating dynamic memory functions. c. Chalkboard/Whiteboard for illustrating memory management.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is static memory allocation? - Why do we need dynamic memory allocation in certain programs? - Can you think of examples where dynamic memory allocation is necessary? **b. Introduction to Dynamic Memory Allocation:** - Define dynamic memory allocation: "It is the process of allocating memory storage during runtime using pointers and specific functions." - Discuss the importance of managing memory efficiently. ***Video Reference***: Introduction to Dynamic Memory Allocation in C **2. Development (30 minutes)** **a. Using malloc() and calloc() (10 minutes)** - **Overview:** Explain the differences between malloc() and calloc() in allocating memory dynamically. - **Examples**: Show simple examples of allocating an array of integers.

	Video Reference: Dynamic Memory Allocation using malloc() and calloc() **b. Resizing with realloc() (10 minutes)** - **Definition**: Explain how realloc() is used to resize dynamically allocated memory. - **Examples**: Demonstrate memory resizing using realloc(). ***Video Reference***: Memory Resizing using realloc() - **c. Memory Deallocation with free() (10 minutes)** - **Overview**: Discuss the importance of releasing unused memory to avoid memory leaks. - **Examples:** Show how to use free() to deallocate memory. ***Video Reference***: Memory Deallocation in C with free() **3. Exercise (5 minutes)** - Activity: Have students write a program that dynamically allocates memory for an array and resizes it. - Example Task: Create a program that asks for user input to dynamically allocate and later free memory for an integer array.
Closure	**1. Summarize key points:** - The purpose and methods of dynamic memory allocation in C programming. - Differentiate between static and dynamic allocation. - Importance of freeing memory to prevent memory leaks. **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 9: Dynamic Memory Allocation, pp. 256-285 **Book 2: "Programming with C" by Byron Gottfried** Chapter 12: Dynamic Memory Allocation, pp. 324-345 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 7: Input/Output, pp. 183-210

Evaluation	**1. Reflective Questions:** - How does dynamic memory allocation improve the flexibility of C programs? - What are the potential risks of using dynamic memory without proper management? **2. Practice Question:** - Write a C program that dynamically allocates memory for a string, resizes it using realloc(), and frees the memory after use.

Lesson Plan No. 5.3	Course Name: Programming in C Topic: File Management in C	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand file handling concepts in C. b. Open, read, write, and close files using standard I/O functions c. Differentiate between text and binary files. d. Use file functions like fopen(), fclose(), fprintf(), fscanf().
Teaching Aids (if any)	a. Slides demonstrating file handling functions. b. Chalkboard/Whiteboard for showing code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is the purpose of file handling in C programming? - How do you think text files differ from binary files? - Have you ever used files in other programming languages? **b. Introduction to File Management:** - Define file management: "File handling in C enables data storage and retrieval on a permanent storage device." - Discuss the significance of reading and writing files in practical applications. ***Video Reference***: File Handling in C **2. Development (30 minutes)** **a. Opening and Closing Files (10 minutes)** - Overview: Explain how to use fopen() and fclose() to open and close files. - Examples: Show opening a text file for reading and writing. ***Video Reference***: Opening and Closing Files in C **b. Reading and Writing Files (10 minutes)** - Definition: Explain how to use fprintf() and fscanf() for text file operations.

	Examples: Demonstrate reading and writing to a file. ***Video Reference***: Reading and Writing Files in C **c. Binary File Handling (10 minutes)** - Overview: Discuss how binary files differ from text files and how to use fwrite() and fread() for binary file operations. - Examples: Show binary file reading and writing. ***Video Reference***: Handling Binary Files in C **3. Exercise (5 minutes)** - Activity: Have students write a program to create a file, write data, and then read the data back. - Example Task: Create a program that reads user input, stores it in a file, and reads it back to display on the screen.
Closure	**1. Summarize key points:** - File management and the importance of persistent data storage. - Differentiation between text and binary files. - Recap of common file handling functions like fopen(), fprintf(), and fread(). **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 11: File Management, pp. 345-376 **Book 2: "Programming with C" by Byron Gottfried** Chapter 13: File I/O, pp. 358-380
Evaluation	**1. Reflective Questions:** - What are some advantages of binary files over text files? - How can improper file handling affect a program? **2. Practice Question:** - Write a C program that reads from a text file and counts the number of characters, words, and lines.

Lesson Plan No. 5.4	Course Name: Programming in C Topic: Pointer Variables and Their Importance	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of pointer variables in C. b. Explain the importance of pointers for efficient memory management. c. Utilize pointer variables to perform various operations. d. Apply pointers in arrays and functions.
Teaching Aids (if any)	a. Slides illustrating pointer variables and their usage b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What are normal variables in C, and how are they stored in memory? - How do pointers differ from regular variables? - Can you think of how pointers might optimize memory usage? **b. Introduction to Pointer Variables:** - Define pointer variables: "A pointer is a variable that holds the memory address of another variable." - Discuss why pointers are crucial for handling dynamic memory and function arguments. ***Video Reference***: Understanding Pointer Variables in C **2. Development (30 minutes)** **a. Declaration and Initialization of Pointers (10 minutes)** - Overview: Explain how to declare and initialize pointer variables. - Examples: Demonstrate creating a pointer to an int and accessing its value.

	Video Reference: Pointer Declaration and Initialization in C **b. Pointer Operations (10 minutes)** - Definition: Explain how pointers can be used to perform various operations like accessing and modifying values. - Examples: Show how to increment pointers and use them with arrays. ***Video Reference:*** Pointer Operations in C **c. Pointer and Functions (10 minutes)** - Overview: Discuss how pointers can be passed to functions to modify arguments. - Examples: Show examples of passing arrays using pointers. ***Video Reference***: Pointers with Functions in C **3. Exercise (5 minutes)** - Activity: Have students write a program that uses a pointer to modify the value of a variable passed to a function. - Example Task: Create a program that uses a pointer to reverse an array of integers.
Closure	**1. Summarize key points:** - The importance of pointer variables in memory management. - The role of pointers in passing arguments to functions. - Recap common operations performed using pointers in arrays and functions. **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 7: Pointer Variables, pp. 181-210 **Book 2: "Programming with C" by Byron Gottfried** Chapter 8: Pointers and Functions, pp. 265-287 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 5: Pointers, pp. 130-150

Evaluation	**1. Reflective Questions:** - How do pointers help in managing arrays and passing arguments to functions? - What are the risks associated with improper use of pointers? **2. Practice Question:** - Write a C program that swaps two integer values using pointer variables.

Lesson Plan No. 5.5	Course Name: Programming in C Topic: Pointer Arithmetic	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of pointer arithmetic in C. b. Perform arithmetic operations on pointers. c. Explain how pointer arithmetic relates to arrays. d. Apply pointer arithmetic in various programming scenarios.
Teaching Aids (if any)	a. Slides illustrating pointer arithmetic concepts. b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What are some arithmetic operations you can perform on regular variables? - Have you ever worked with arrays using pointers in C? - Why might pointer arithmetic be useful when working with arrays? **b. Introduction to Pointer Arithmetic:** - Define pointer arithmetic: "Pointer arithmetic involves performing operations like addition and subtraction on pointers, which allow you to navigate through memory locations." - Discuss how pointer arithmetic simplifies working with arrays and other data structures. ***Video Reference***: Introduction to Pointer Arithmetic in C **2. Development (30 minutes)** **a. Basics of Pointer Arithmetic (10 minutes)** - Overview: Explain the basics of pointer arithmetic, including addition, subtraction, and comparison of pointers. - Examples: Show how to increment a pointer to traverse an array. ***Video Reference***: Basics of Pointer Arithmetic in C **b. Pointer Arithmetic with Arrays (10 minutes)** - Definition: Discuss how pointer arithmetic is related to arrays

	and how it can be used to iterate through array elements. - Examples: Demonstrate accessing array elements using pointer arithmetic. ***Video Reference***: Pointer Arithmetic and Arrays in C **c. Pointer Subtraction and Comparison (10 minutes)** - Overview: Explain how to subtract pointers and compare them to find the difference in memory locations. - Examples: Show examples of pointer subtraction and comparison. ***Video Reference***: Pointer Subtraction and Comparison in C **3. Exercise (5 minutes)** - Activity: Have students write a program that uses pointer arithmetic to reverse an array of integers. - Example Task: Create a program that iterates through an array using pointer arithmetic and prints each element.
Closure	**1. Summarize key points:** - The concept and importance of pointer arithmetic in navigating memory. - The relationship between pointers and arrays. - Recap common pointer arithmetic operations and their practical applications. **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 7: Pointer Arithmetic, pp. 211-230 **Book 2: "Programming with C" by Byron Gottfried** Chapter 8: Pointers and Arrays, pp. 287-305 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 5: Pointers and Arrays, pp. 150-172

Evaluation	**1. Reflective Questions:** - How does pointer arithmetic help in accessing and manipulating arrays? - What precautions should be taken when performing arithmetic on pointers? **2. Practice Question:** - Write a C program that uses pointer arithmetic to sum the elements of an integer array.

Lesson Plan No. 5.6	Course Name: Programming in C Topic: Passing Parameters by Reference	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of passing parameters by reference in C. b. Explain the difference between passing by value and passing by reference. c. Use pointers to pass parameters by reference. d. Apply passing by reference in practical scenarios such as modifying function arguments.
Teaching Aids (if any)	a. Slides illustrating passing parameters by reference. b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What happens when you pass a variable by value to a function? - Have you ever needed to modify a variable within a function and keep the changes outside the function? - Why do you think passing by reference might be useful? **b. Introduction to Passing Parameters by Reference:** - Define passing by reference: "Passing by reference involves passing the memory address of a variable to a function, allowing the function to modify the original variable." - Discuss the advantages of passing by reference over passing by value. ***Video Reference***: Introduction to Passing Parameters by Reference in C **2. Development (30 minutes)** **a. Passing by Value vs. Passing by Reference (10 minutes)** - Overview: Explain the differences between passing by value and passing by reference. - Examples: Show examples of both methods with simple functions.

	Video Reference: Passing by Value vs. Passing by Reference in C **b. Using Pointers for Reference Passing (10 minutes)** - Definition: Discuss how pointers are used to pass parameters by reference. - Examples: Demonstrate a function that modifies an argument using a pointer. ***Video Reference***: Using Pointers for Passing Parameters by Reference in C **c. Practical Applications (10 minutes)** - Overview: Discuss scenarios where passing by reference is essential, such as swapping values and modifying arrays. - Examples: Show a function that swaps two integers using passing by reference. ***Video Reference***: Practical Applications of Passing by Reference in C **3. Exercise (5 minutes)** - Activity: Have students write a program that uses a function to swap two integer values using passing by reference. - Example Task: Create a program that takes two integers as input and swaps their values using a function that accepts pointers.
Closure	**1. Summarize key points:** - The concept and importance of passing parameters by reference. - The role of pointers in passing by reference. - Recap scenarios where passing by reference is more efficient than passing by value. **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 6: Passing Parameters by Reference, pp. 164-180 **Book 2: "Programming with C" by Byron Gottfried** Chapter 7: Functions and Parameter Passing, pp. 243-260 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 4: Functions and Pointers, pp. 110-129

Evaluation	**1. Reflective Questions:** - Why is passing by reference more efficient in certain cases? - How can passing by reference prevent unnecessary copying of data? **2. Practice Question:** - Write a C program that calculates the sum of two integers using passing by reference.

Lesson Plan No. 5.7	Course Name: Programming in C Topic: Pointer to Pointer	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of pointer to pointer in C. b. Explain how a pointer can store the address of another pointer. c. Use double pointers in practical programming scenarios. d. Apply pointer to pointer in dynamic memory allocation and multi-dimensional arrays.
Teaching Aids (if any)	a. Slides illustrating pointer to pointer concepts. b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What is a pointer, and how does it store the address of a variable? - Have you ever needed to store the address of a pointer? - Why do you think having a pointer to another pointer could be useful? **b. Introduction to Pointer to Pointer:** - Define pointer to pointer: "A pointer to pointer is a pointer that stores the address of another pointer, which in turn stores the address of a variable." - Discuss the use cases of pointer to pointer, such as dynamic memory allocation and handling arrays of pointers. ***Video Reference***: Introduction to Pointer to Pointer in C **2. Development (30 minutes)** **a. Declaration and Usage of Double Pointers (10 minutes)** - Overview: Explain how to declare and use double pointers. - Examples: Show how to create a double pointer and access the value it points to. ***Video Reference***: Double Pointers in C

	b. Pointer to Pointer in Dynamic Memory Allocation (10 minutes) - Definition: Discuss how double pointers are used in dynamic memory allocation, especially for multi-dimensional arrays. - Examples: Demonstrate dynamic allocation of a 2D array using double pointers. ***Video Reference***: Pointer to Pointer in Dynamic Memory Allocation **c. Pointer to Pointer in Multi-Dimensional Arrays (10 minutes)** - Overview: Explain how to use double pointers to navigate through multi-dimensional arrays. - Examples: Show examples of accessing elements in a 2D array using pointer to pointer. ***Video Reference***: Using Pointer to Pointer in Multi-Dimensional Arrays in C **3. Exercise (5 minutes)** - Activity: Have students write a program that uses pointer to pointer to dynamically allocate memory for a 2D array. - Example Task: Create a program that allocates memory for a 2D integer array using double pointers and prints its elements.
Closure	**1. Summarize key points:** - The concept of pointer to pointer and how it works. - Practical applications of double pointers in dynamic memory allocation and multi-dimensional arrays. - Recap of how double pointers can be used to store and manipulate addresses of other pointers. **2. Suggested Readings** **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 8: Pointers, pp. 245-266 **Book 2: "Programming with C" by Byron Gottfried** Chapter 10: Pointers, Arrays, and Memory, pp. 326-344

Evaluation	1. **Reflective Questions**: - How does a pointer to a pointer differ from a regular pointer? - In what scenarios would you prefer to use a pointer to a pointer over a single pointer? 2. **Practice Question**: - Write a C program that dynamically allocates memory for a 2D array using a pointer to a pointer, assigns values, and prints the array elements.

Lesson Plan No. 5.8	Course Name: Programming in C Topic: Pointer to Functions	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of function pointers in C. b. Declare and use function pointers. c. Explain how function pointers can be passed as arguments to other functions. d. Apply function pointers in scenarios such as call backs and dynamic function calls.
Teaching Aids (if any)	a. Slides illustrating pointer to function concepts. b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - Have you ever needed to call different functions dynamically in a program? - Why do you think it might be useful to pass a function as an argument to another function? b. **Introduction to Function Pointers**: - Define function pointers: "A function pointer is a pointer that stores the address of a function and can be used to invoke that function." - Discuss how function pointers allow dynamic function calls, especially in cases like callbacks. ***Video Reference***: Introduction to Function Pointers in C **2. Development (30 minutes)** **a. Declaring and Using Function Pointers (10 minutes)** - Overview: Explain how to declare and use function pointers. - Examples: Show examples of creating and using function pointers to invoke a function. ***Video Reference***: Basics of Function Pointers in C b. **Passing Function Pointers as Arguments (10 minutes)** - Definition: Discuss how function pointers can be passed as arguments to other functions, especially in callback scenarios. - Examples: Demonstrate using function pointers in a callback

	mechanism. ***Video Reference***: Function Pointers as Arguments in C c. **Practical Applications (10 minutes)** - Overview: Explore scenarios where function pointers are essential, such as dynamic function selection, event-driven programming, and signal handling. - Examples: Create a program where function pointers are used to select and call different mathematical operations. ***Video Reference***: Practical Applications of Function Pointers in C **3. Exercise (5 minutes)** - **Activity**: Have students write a program that uses function pointers to call different mathematical functions based on user input. - **Example Task**: Create a menu-driven program that allows users to select an operation (addition, subtraction, multiplication) using a function pointer.
Closure	1. **Summarize key points**: - The concept of function pointers and how they work. - The importance of using function pointers in callbacks and dynamic function calls. - Recap of how function pointers are declared, used, and passed as arguments. 2. **Suggested Readings**: **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 9: Function Pointers, pp. 272-290 **Book 2: "Programming with C" by Byron Gottfried** Chapter 12: Functions and Pointers, pp. 379-395 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 6: Function Pointers, pp. 210-225
Evaluation	1. **Reflective Questions**: - How do function pointers enhance flexibility in function selection and execution? - What are the advantages of using function pointers in callback mechanisms? 2. **Practice Question**: - Write a C program that uses function pointers to perform

	mathematical operations (addition, subtraction, multiplication) based on user input.

Lesson Plan No. 5.9	Course Name: Programming in C Topic: Dangling Pointer	Course No.: BCAMJ-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand what a dangling pointer is and why it occurs. b. Explain how dangling pointers lead to undefined behaviour. c. Apply strategies to avoid dangling pointers in programs. d. Identify and debug programs with dangling pointer issues.
Teaching Aids (if any)	a. Slides illustrating dangling pointer scenarios b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - What happens when you delete or free a dynamically allocated variable? - Can you think of any issues that might occur when a pointer is left pointing to a memory location that has been freed? **b. Introduction to Dangling Pointers**: - Define dangling pointer: "A dangling pointer occurs when a pointer still points to a memory location that has been freed or deleted, leading to undefined behavior." - Discuss how dangling pointers can cause errors in programs. ***Video Reference***: Introduction to Dangling Pointers in C **2. Development (30 minutes)** **a. Causes of Dangling Pointers (10 minutes)** - Overview: Explain the common causes of dangling pointers, such as freeing memory or returning local variables from functions. - Examples: Show how dangling pointers occur with dynamically allocated memory. ***Video Reference***: Common Causes of Dangling Pointers **b. Avoiding Dangling Pointers (10 minutes)** - Definition: Discuss best practices to avoid dangling pointers, such as setting pointers to NULL after freeing them. - Examples: Demonstrate how to avoid dangling pointers in practical programming scenarios.

	Video Reference: How to Avoid Dangling Pointers **c. Debugging Dangling Pointer Issues (10 minutes)** - Overview: Explore strategies for identifying and debugging dangling pointer issues, such as using debugging tools like Valgrind. - Examples: Show examples of debugging programs with dangling pointer problems. ***Video Reference***: Debugging Dangling Pointer Issues in C **3. Exercise (5 minutes)** - **Activity**: Have students write a program that dynamically allocates memory, frees it, and correctly handles pointers to avoid dangling pointers. - **Example Task**: Create a program that allocates memory for an array, frees the memory, and ensures that all pointers to the memory are handled correctly.
Closure	1. **Summarize key points**: - The concept of dangling pointers and their causes. - Strategies to avoid dangling pointers in C programs. - Recap of how dangling pointer issues can lead to undefined behavior and how to debug them. 2. **Suggested Readings**: **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 10: Dynamic Memory Allocation, pp. 315-335 **Book 2: "Programming with C" by Byron Gottfried** Chapter 11: Memory Management, pp. 365-380 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 8: Memory Management, pp. 230-250
Evaluation	1. **Reflective Questions**: - How can dangling pointers lead to undefined behavior in programs? - What strategies can be used to avoid dangling pointers? 2. **Practice Question**: - Write a C program that demonstrates a dangling pointer scenario and corrects it by setting the pointer to NULL after freeing the memory.

Lesson Plan No. 5.10	Course Name: Programming in C Topic: Dynamic Memory Allocation	Course No.: COM-101

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of dynamic memory allocation in C. b. Use the `malloc`, `calloc`, `realloc`, and `free` functions. c. Explain the difference between static and dynamic memory allocation. d. Apply dynamic memory allocation in real-world programs.
Teaching Aids (if any)	a. Slides illustrating dynamic memory allocation concepts. b. Chalkboard/Whiteboard for code examples.
Teaching Development	**1. Introduction (5 minutes)** **a. Pre-Discussion Questions:** - Have you ever needed to create variables or arrays where the size isn't known at compile time? - Why do you think dynamic memory allocation might be useful in such cases? b. **Introduction to Dynamic Memory Allocation**: - Define dynamic memory allocation: "Dynamic memory allocation is the process of allocating memory during runtime using library functions like `malloc`, `calloc`, and `realloc`." - Discuss the importance of dynamic memory allocation in efficient memory management. ***Video Reference***: Introduction to Dynamic Memory Allocation in C **2. Development (30 minutes)** **a. Dynamic Memory Allocation vs. Static Memory Allocation (5 minutes)** - **Overview**: Discuss the differences between static memory allocation (at compile time) and dynamic memory allocation (at runtime). - **Examples**: Compare scenarios where static memory allocation would fail or be inefficient, and how dynamic memory allocation solves these problems. ***Video Reference***: Static vs. Dynamic Memory Allocation in C **b. Real-world Applications of Dynamic Memory Allocation (5 minutes)**

	- **Overview**: Explore real-world scenarios where dynamic memory allocation is useful, such as managing large datasets, flexible data structures (like linked lists, trees, and graphs), and memory management in operating systems. - **Examples**: Discuss how dynamic memory allocation is used in applications like dynamic arrays, queues, and stacks. ***Video Reference**:* Real-world Use Cases of Dynamic Memory Allocation in C **3. Exercise (5 minutes)** - **Activity**: Have students write a C program that dynamically allocates memory for an array, fills the array with user input, and resizes the array using `realloc`. - **Example Task**: Create a program that allows the user to input a number of integers, stores them dynamically, and then resizes the memory to store additional integers.
Closure	1. **Summarize key points**: - The concept and importance of dynamic memory allocation in C programming. - The use of `malloc`, `calloc`, `realloc`, and `free` to manage memory dynamically. - The difference between static and dynamic memory allocation and when to use each. 2. **Suggested Readings**: **Book 1: "Programming in ANSI C" by E. Balagurusamy** Chapter 10: Dynamic Memory Allocation, pp. 315-335 **Book 2: "Programming with C" by Byron Gottfried** Chapter 12: Dynamic Memory and Pointers, pp. 395-412 **Book 3: "C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie** Chapter 8: Dynamic Memory Management, pp. 230-250
Evaluation	1. **Reflective Questions**: - What are the benefits of dynamic memory allocation compared to static memory allocation? - Why is it important to free dynamically allocated memory after its use? 2. **Practice Question**: - Write a C program that allocates memory dynamically for a list of integers, accepts user input, resizes the memory using `realloc`, and finally deallocates the memory using `free`.