C PROGRAMMING

COURSE CODE: BCA-102

YEAR/SEMESTER: I/I **CREDIT: 3**

WORKLOAD: 6 Hrs/ WEEK (THEORY: 3 Hrs, PRACTICAL: 3 Hrs)

Course description:

This course provides a comprehensive introduction to the C programming language, a foundational tool in computer science and software development. Emphasizing structured and procedural programming techniques, the course covers the core concepts of C including variables, data types, operators, control structures, arrays, functions, pointers, structures, unions, and file handling. Students will gain practical skills in writing efficient, modular, and error-free code, reinforced through hands-on lab sessions. The course is designed to develop algorithmic thinking, problem-solving ability, and a strong understanding of how memory and low-level operations work in modern computing environments. This course provides the essential groundwork for more advanced in programming for software development, algorithms, and systems programming.

Course objectives:

The main objective of this course to provide students both the theoretical foundation and practical knowledge of programming using C programming language. After the completion of the course, students will be able to:

- Understand the basic structure and syntax of the C programming language, including data types, operators, and control statements.
- Develop algorithmic thinking and structured programming techniques to solve computational problems.
- Apply modular programming concepts using functions to enhance code reusability and clarity.
- Manipulate arrays, strings, and pointers effectively for data processing and memory management.
- Use structures and unions to model and organize complex data in C.
- Perform basic file input and output operations to read, write, and manage data in text and binary files.

- Write, compile, debug, and test C programs using standard development tools and environments.
- Establish a strong foundation for learning advanced programming topics, such as data structures, operating systems, and embedded systems.

Course contents

1 Unit-1 Introduction to C programming language and basics **5 Hrs.**

1.1 Evolution of programming languages

1.2 History, characteristics and applications of C.

1.3 Structure of a C program.

1.4 Compilation and execution process of a C program.

1.5 C Tokens: Keywords, identifiers, constants, string literals, operators.

1.6 Variables: Declaration, initialization, scope, and lifetime.

1.7 Data Types: Basic data types (int, char, float, double, void), derived data types, user-defined data types.

1.8 Type Casting: Implicit and explicit type conversion.

1.9 Operators and Expressions: Arithmetic, relational, logical, bitwise, assignment, increment/decrement, conditional (ternary), special operators (sizeof, comma, address-of, dereference). Operator precedence and associativity.

Unit 2: Input/Output and Control Structures **11 Hrs**

2.1 Standard Input and Output Functions:

2.1.1 Formatted I/O: printf() and scanf().

2.1.2 Unformatted I/O Functions: Character I/O (getchar(), putchar(), getch(), getche(), putch()),

2.2 Control Structures:

2.3 Decision Making Statements:

2.3.1 if statement

2.3.2 if-else statement

2.3.3 Nested if-else statement

2.3.4 else-if ladder

2.3.5 switch statement

2.3.6 Conditional operator (? :)

2.4 Looping Statements:

2.4.1 while loop

2.4.2 do-while loop

2.4.3 for loop

2.4.4 Nested loops

2.5 Jump Statements:

2.5.1 break statement

2.5.2 continue statement

2.5.3 goto statement

Unit 3: Functions, Arrays and String **11 Hrs**

3.1 Functions:

3.1.1 Introduction to functions: Advantages, function definition, function declaration/prototype,

3.1.2 function call

3.1.3 Types of functions: Library functions vs. User-defined functions

3.1.4 Function arguments: Call by value, Call by reference

3.1.5 Recursion: Concepts and examples

3.2 Arrays:

3.2.1 Introduction to arrays: Declaration, initialization, accessing elements

3.2.2 One-dimensional arrays: Processing and examples

3.2.3 Two-dimensional arrays: Declaration, initialization, accessing elements (matrix operations).

3.2.4 Multi-dimensional arrays

3.2.5 Arrays and functions

3.3 Strings:

3.3.1 Introduction to strings: Declaration, initialization

3.3.2 String input/output(gets,puts)

3.3. 3 String handling functions(strlen,strrev,strcpy,strupr,strlwr,strcmp,strcat)

3.3.4 Array of strings.

Unit 4: Structures, Unions, and Enumerations **5 hrs**

4.1 Structures and Unions:

Defining and declaring structures.

Accessing members, including nested structures, arrays of structures.

Passing structures to functions.

Defining and declaring unions; comparison with structures.

4.2 Enumerations (Enums) and typedef implementation .

Unit 5: Pointers and Memory Management **5 hrs**

5.1 Pointer declaration, initialization, and dereferencing

5.1.2 Pointer arithmetic

5.1.3 Pointers with arrays, string, function, structure.

5.2 Dynamic memory allocation: malloc, calloc, realloc, free

5.3 DMA with structure

5.4 Dangling pointers and memory leaks

Unit 6: File Handling, Command-Line Arguments, and Graphics **11 Hrs**

6.1 File Handling:

6.1.2 Concepts of files (text vs. binary) and file operations (opening, closing, reading, writing).

6.1.3 File modes and standard I/O functions (fgetc, fputc, fgets, fputs, fprintf, fscanf, fread, fwrite).

6.1.4 Random access (fseek, ftell, rewind) and error handling.

6.2 Command-Line Arguments:

6.2.1 Introduction to main() function parameters (argc, argv).

6.2.2 Passing and accessing arguments from the command line.

6.3 Graphics in C:

6.3.1 Introduction to graphics programming (basic concepts, graphics modes).

6.3.2 Drawing primitives: Points, lines, and common geometric shapes (rectangles, circles, ellipses, arcs, polygons).

6.3.3 Managing colors and fill styles for graphical output.

6.3.4 Displaying and formatting text in graphics mode.

6.3.5 Fundamentals of simple animation (clearing the screen, redrawing elements, using delays).

Laboratory work **[48 hrs.]**

The practical component will reinforce theoretical concepts through hands-on programming exercises. A minimum of 48 hours (3 hours/week for 16 weeks) will be dedicated to practical sessions. Students are expected to complete assignments on a weekly basis.

General Instructions for Practical Sessions:

- Familiarization with a C IDE (e.g., Code::Blocks, Visual Studio Code with GCC, Eclipse CDT, Dev-C++, Turbo C++).
- Understanding the compilation and execution process.
- Debugging techniques.
- Emphasis on practical application for complex topics, especially Pointers and File Handling, to ensure thorough understanding.

List of Practical Concepts to implement (with enhanced emphasis):

1. Fundamental Programming Constructs:
   - Implement programs demonstrating basic input/output, arithmetic operations, and data type conversions.
   - Apply various operators (arithmetic, relational, logical, bitwise, conditional) in expressions.
2. Control Flow Mechanisms (Practical Emphasis):
   - Gain practical proficiency in utilizing conditional statements (if, if-else, switch) to control program flow based on various conditions.
   - Implement and analyze iterative solutions using while, do-while, and for loops for repetitive tasks.
   - Demonstrate the effective use of jump statements (break, continue, goto) for specific control transfers.
3. Functions and Modularity (Practical Emphasis):
   - Design and implement modular programs using user-defined functions, focusing on effective argument passing (call by value, call by reference) and return types.
   - Explore practical applications of recursive problem-solving approaches.
4. Array and String Manipulation (Practical Emphasis):
   - Gain hands-on experience performing various operations on one-dimensional, two-dimensional, and multi-dimensional arrays, including searching, sorting, and matrix operations.
   - Implement practical solutions involving string input/output and proficiently utilize standard string handling functions.
   - Work effectively with arrays of strings for real-world data representation.
5. Pointers and Dynamic Memory (Strong Practical Emphasis):

- Develop robust programs demonstrating core pointer concepts, including declaration, initialization, and dereferencing.
- Master pointer arithmetic and deeply understand the relationship between pointers and arrays/strings for efficient memory access.
- Gain comprehensive practical experience in dynamic memory management using malloc(), calloc(), realloc(), and free() for flexible data structures.

6. Structured Data Types:
   - Define and manipulate structures, including nested structures, arrays of structures, and pointers to structures.
   - Utilize unions and understand their memory implications.
   - Implement programs using enumerated types.
7. File Operations (Practical Emphasis):
   - Achieve practical competence in performing fundamental file operations (create, open, read, write) on both text and binary files.
   - Implement formatted and block I/O operations for efficient data handling with files.
   - Navigate and manipulate file pointers for random access within files.
   - Effectively incorporate error handling mechanisms for reliable file operations.
8. Command-Line Interaction:
   - Develop programs that accept and process command-line arguments.
9. Basic Graphics Programming:
   - Initialize graphics mode and draw fundamental shapes like points, lines, rectangles, circles, and ellipses.
   - Experiment with colors and fill patterns.
   - Display and format text within graphics output.
   - Implement simple animation sequences.

Required readings

Balagurusamy, E. (n.d.). *Programming in ANSI C* (Latest ed.). Tata McGraw-Hill Education.

Deitel, P. J., & Deitel, H. M. (n.d.). *C how to program* (Latest ed.). Pearson Education.

Kernighan, B. W., & Ritchie, D. M. (n.d.). *The C programming language* (Latest ed.). Prentice Hall.

Prata, S. (n.d.). *C primer plus* (Latest ed.). Pearson Education.