B. Sc. Part – I Semester – I

COMPUTER SCIENCE (Optional)

PROBLEM SOLVING USING COMPUTERS

Question Bank

Concept of problem solving-

1. **Understanding the Problem:**
   - What is the main objective of the problem?
   - Can you identify any specific inputs and outputs required for the problem?
   - Are there any constraints or limitations that need to be considered?
   - Can you explain the problem statement in your own words?
   - What are the inputs required for this problem? How are they provided?
   - Can you identify any potential edge cases or special conditions to consider?
2. **Breaking Down the Problem:**
   - How would you decompose the main problem into smaller, more manageable tasks?
   - Can you identify any repetitive tasks that can be modularized into functions?
   - What data structures or algorithms might be useful in solving the problem?
3. **Developing a Plan:**
   - What approach or algorithm would you use to solve the problem?
   - Can you outline the steps or pseudo-code needed to implement your solution?
   - How would you handle any edge cases or exceptional scenarios?
   - What algorithm or approach would you use to solve this problem?
   - Can you outline the steps you'll take to implement your solution?
   - How would you handle errors or invalid inputs during execution?
   - What algorithm or approach would you use to solve this problem?
   - Can you outline the steps you'll take to implement your solution?
   - How would you handle errors or invalid inputs during execution?
4. **Implementing the Plan:**
   - How would you translate your plan into actual C code?
   - Are there any specific language features or libraries that would be helpful in the implementation?
   - How would you test each component of your code to ensure it behaves as expected?
5. **Testing and Debugging:**
   - What test cases would you use to verify the correctness of your code?
   - How would you handle runtime errors or unexpected behaviors during testing?
   - Can you identify any potential sources of bugs and how you would debug them?
   - How would you verify that your program works correctly?
   - What test cases would you use to ensure that all possible scenarios are covered?
   - If your program doesn't work as expected, how would you debug it?

6. **Evaluating and Iterating:**
   - After implementing your solution, how would you evaluate its performance and efficiency?
   - Are there any ways you could optimize your code further?
   - What would you do differently if you were to solve a similar problem in the future?
   - How would you assess the efficiency and performance of your solution?
   - Are there any optimizations you could make to improve your program's performance?
   - What would you do differently if you were to solve this problem again?

Problem definition-

1. **Modularization:**
   - How would you break down a large program into smaller, more manageable modules?
   - Can you explain the benefits of modular programming in C?
   - What criteria would you use to decide when to create a new module or function?

2. **Abstraction and Encapsulation:**
    - **How would you use abstraction to hide implementation details and expose only necessary interfaces?**
    - **Can you give examples of how you would encapsulate data and behavior within C structures and functions?**
    - **What are the advantages of encapsulation in C program design?**
3. **Code Reusability:**
    - **How would you design your C programs to promote code reusability?**
    - **Can you explain the difference between function-level and module-level code reuse?**
    - **What strategies would you use to write reusable code in C?**
4. **Scalability and Maintainability:**
    - **How would you design your C programs to be scalable as the requirements change?**
    - **What techniques would you use to ensure your C codebase is maintainable by yourself and others?**
    - **Can you discuss the trade-offs between simplicity and flexibility in C program design?**
5. **Error Handling and Robustness:**
    - **How would you design your C programs to handle errors gracefully?**
    - **Can you describe strategies for robust input validation and error recovery?**
    - **What role do assertions and error codes play in designing robust C programs?**
6. **Performance Considerations:**
    - **How would you optimize your C programs for performance?**
    - **Can you discuss the impact of algorithm choice, data structure selection, and memory management on program performance?**
    - **What tools and techniques would you use to profile and optimize your C code?**

Debugging

1. **Understanding the Bug:**
    - **Can you describe the unexpected behavior or error you're encountering?**
    - **What part of the code do you suspect is causing the issue?**
    - **Have you observed any patterns or conditions under which the bug occurs?**
2. **Identifying the Cause:**
    - **Have you checked for syntax errors or typos in the code?**
    - **Are you using any external libraries or functions that might be causing conflicts?**
    - **Have you examined the values of variables and data structures to identify any inconsistencies?**
3. **Isolating the Issue:**
    - **Can you reproduce the bug consistently, or does it occur sporadically?**
    - **Have you tried narrowing down the problem by commenting out sections of code or using print statements?**
    - **Are there specific inputs or conditions that trigger the bug?**
4. **Using Debugging Tools:**
    - **Have you utilized a debugger tool like GDB to step through the code and inspect variable values?**
    - **Are you familiar with using printf statements or logging to track the program's execution flow?**
    - **Have you considered using memory debugging tools like Valgrind to detect memory-related issues?**
5. **Analyzing Error Messages:**

- What error messages or warnings are being generated by the compiler or runtime environment?
- Do these error messages provide any clues about the source of the problem?
- Have you searched for similar error messages online or in documentation to find possible solutions?

6. Testing and Verification:
   - Have you created a minimal reproducible example to isolate the bug?
   - Can you write test cases to verify that the bug has been fixed once you identify the cause?
   - Are you testing your code on different platforms or environments to ensure compatibility?
7. Seeking Help:
   - Have you asked colleagues or peers to review your code and provide fresh perspectives?
   - Are there online forums or communities where you can seek advice or assistance?
   - Have you considered consulting official documentation or resources related to the tools or libraries you're using?

Types of errors in programming

1. Syntax Errors:
   - What are syntax errors in C programming?
   - Can you provide examples of common syntax errors?
   - How does the compiler handle syntax errors, and how can programmers identify and fix them?
2. Logical Errors:
   - What distinguishes logical errors from syntax errors?
   - Can you give an example of a logical error in C code?
   - How do programmers typically debug and fix logical errors?
3. Runtime Errors:
   - What are runtime errors, and when do they occur?
   - Can you explain common causes of runtime errors in C programs?
   - How can programmers prevent or handle runtime errors in their code?
4. Semantic Errors:
   - What do semantic errors represent in programming?
   - Can you provide examples of semantic errors in C code?
   - How do semantic errors differ from other types of errors, and how can programmers identify and correct them?
5. Linker Errors:
   - What are linker errors, and when do they occur?
   - Can you explain scenarios that lead to linker errors in C programming?
   - How can programmers resolve linker errors in their programs?
6. Compilation Errors:
   - What is the compilation process in C programming?
   - Can you list some common compilation errors and their causes?
   - How do programmers typically address compilation errors during software development?

Documentation

1. Header Documentation:
   - What information should be included in the header documentation of a C function?
   - How do you properly document the purpose and usage of a function in its header?
   - Can you explain the significance of including parameters, return types, and possible side effects in function documentation?
2. Inline Comments:

- Why are inline comments important in C code?
- How do you decide where to place inline comments within your code?
- Can you provide examples of situations where inline comments are particularly useful?

3. Code Structure and Organization:
   - What strategies can be employed to maintain clear and organized code structure?
   - How should you document the structure and purpose of header files in a project?
   - Can you explain the benefits of using consistent naming conventions and file organization in documentation?
4. Function Documentation:
   - What details should be included in the documentation of a C function?
   - How do you document the parameters, return values, and any error conditions of a function?
   - Can you provide examples of how function documentation can improve code readability and maintainability?
5. Documenting Data Structures:
   - How should you document the design and usage of data structures in C code?
   - What information is essential to include in the documentation of a data structure?
   - Can you explain how documenting data structures contributes to code reusability and comprehension?
6. Generating Documentation from Comments:
   - What tools are available for automatically generating documentation from comments in C code?
   - How do you configure and use these tools to generate documentation for a C project?
   - Can you discuss the advantages and limitations of using automated documentation generation?

Linux Operating System

1. File System and Commands:
   - Explain the hierarchical structure of the Linux file system.
   - What are some commonly used commands for navigating the file system?
   - How do you create, copy, move, and delete files and directories in Linux?
2. Process Management:
   - What is a process in Linux? How is it different from a program?
   - How do you start, stop, pause, and resume processes in Linux?
   - Can you explain the concept of process priorities and how they're managed in Linux?
3. User and Group Management:
   - How do you create, modify, and delete users and groups in Linux?
   - What are the differences between regular users and superusers (root) in Linux?
   - How do you manage permissions to restrict or allow access to files and directories?
4. Networking:
   - Explain how networking is configured in Linux.
   - What commands do you use to check network connectivity, configure network interfaces, and troubleshoot network issues?
   - How do you set up and manage services like SSH, FTP, or web servers in Linux?
5. Package Management:
   - What is package management in Linux, and why is it important?
   - How do you install, update, and remove software packages using package managers like APT or YUM?
   - Can you explain the concept of package repositories and how they're used in Linux?
6. System Administration and Security:
   - What are some common administrative tasks performed in Linux?
   - How do you monitor system performance, check system logs, and manage system services?
   - What security measures can be implemented in Linux to protect against unauthorized access and malicious attacks?
7. Shell Scripting:
   - What is shell scripting, and how is it useful in Linux?
   - How do you write and execute a basic shell script in Linux?

- Can you give examples of common shell scripting tasks, such as file manipulation, process management, or automation?

Introduction to GCC Compiler

1. Overview and Installation:
    - What is GCC, and what does it stand for?
    - Can you explain the role of GCC in the software development process?
    - How do you install GCC on different operating systems like Linux, macOS, and Windows?
2. Compilation Process:
    - What are the main stages of the compilation process in GCC?
    - Can you explain each stage of the compilation process in detail?
    - How does GCC generate machine code from source code?
3. Command-Line Options:
    - What are command-line options, and how are they used with GCC?
    - Can you give examples of common GCC options used for compilation, optimization, and debugging?
    - How do you specify input files, output files, and other parameters when using GCC?
4. Language Support:
    - What programming languages does GCC support for compilation?
    - Can you explain the level of support for each programming language (e.g., C, C++, Fortran)?
    - How do you specify the programming language to be compiled when using GCC?
5. Optimization Techniques:
    - What is compiler optimization, and why is it important?
    - Can you explain some common optimization techniques used by GCC?
    - How do you enable or disable optimization levels when compiling with GCC?
6. Debugging and Profiling:
    - How does GCC support debugging and profiling of programs?
    - What options can be used to generate debugging information in the compiled binary?
    - Can you explain how to use tools like GDB and Valgrind with GCC-compiled programs for debugging and profiling?
7. Cross-Compilation:
    - What is cross-compilation, and when is it necessary?
    - How do you configure GCC for cross-compilation to target different architectures or platforms?
    - Can you explain the challenges and considerations involved in cross-compilation with GCC?
8. GCC Extensions and Compatibility:
    - What are GCC extensions, and how do they differ from standard language features?
    - Can you give examples of GCC-specific language extensions or features?
      How do you ensure code portability when using GCC extensions?

Built-In Standard Library

1. Input/Output (stdio.h):
    - What are the main functions for input and output in C?
    - How do you read input from the user using scanf and fgets? What are their differences?
    - Explain the purpose of printf and its formatting options.
2. String Manipulation (string.h):
    - What functions are available in string.h for manipulating strings?
    - How do you concatenate two strings in C?
    - What is the difference between strcpy and strncpy?
3. Memory Management (stdlib.h):
    - Explain the purpose of dynamic memory allocation functions like malloc, calloc, and realloc.
    - How do you free dynamically allocated memory using free?
    - What is the role of exit function in C?
4. Math Functions (math.h):
    - What mathematical functions are available in math.h?
    - How do you calculate the square root of a number using sqrt?

- Explain the purpose of pow, sin, and cos functions.

5. File Handling (stdio.h, stdlib.h):
   - How do you open a file for reading and writing using fopen?
   - What functions are used for reading and writing data to a file in C?
   - How do you close a file after performing file operations?
6. Character Handling (ctype.h):
   - What functions are provided in ctype.h for character handling?
   - How do you convert characters between uppercase and lowercase using toupper and tolower?
   - Explain the purpose of isdigit, isalpha, and isspace functions.
7. Error Handling (errno.h):
   - What is the purpose of errno.h in C programming?
   - How do you handle errors using errno and perror?
   - Can you explain the use of strerror function?
8. Time and Date Functions (time.h):
   - What functions are available in time.h for working with time and date?
   - How do you retrieve the current time using time function?
   - Explain the purpose of localtime and strftime functions.

C Program Structure

1. Header Files and Preprocessor Directives:
   - What is the purpose of header files in C programming?
   - Can you explain the role of preprocessor directives like #include and #define?
   - How do you include standard library headers versus user-defined headers in a C program?
2. Main Function:
   - What is the significance of the main() function in a C program?
   - What are the valid return types and parameters of the main() function?
   - How do you pass command-line arguments to the main() function?
3. Function Declaration and Definition:
   - What is the syntax for declaring a function in C?
   - How do you define a function, and what are the components of a function definition?
   - Can you explain the concept of function prototypes and their importance?
4. Variable Declaration and Scope:
   - How do you declare variables in C, and what are the rules for naming variables?
   - What is variable scope, and how is it determined in C?
   - Can you explain the difference between local, global, and static variables?
5. Control Structures:
   - What are the basic control structures in C programming?
   - How do you use conditional statements like if, else if, and switch in C?
   - Can you describe the syntax and use of loop structures such as for, while, and do-while?
6. Comments and Documentation:
   - What is the purpose of comments in a C program?
   - How do you write single-line and multi-line comments in C?
   - Can you explain the importance of documenting code for readability and maintainability?
7. Program Organization and Modularity:
   - How do you organize a C program into multiple source files?
   - What are header files, and how do they facilitate modular programming?
   - Can you describe the concept of separate compilation and linking in C?

C Language-

1. Algorithm & Flowchart:
   1. **Understanding Algorithms:**

- **What is an algorithm, and why is it important in programming?**
- **Can you explain the difference between an algorithm and a program?**
- **How do you analyze the efficiency of an algorithm, and what factors contribute to its complexity?**

2. **Designing Algorithms:**
   - **How do you approach designing an algorithm to solve a specific problem?**
   - **Can you describe the process of breaking down a problem into smaller subproblems?**
   - **What are some common algorithm design techniques, such as divide and conquer, dynamic programming, or greedy algorithms?**
3. **Implementing Algorithms in C:**
   - **How do you translate an algorithm into C code?**
   - **What data structures and control structures are commonly used in implementing algorithms in C?**
   - **Can you provide examples of C code for common algorithms, such as sorting, searching, or graph traversal?**
4. **Flowcharts:**
   - **What is a flowchart, and how is it useful in algorithm design?**
   - **Can you explain the basic symbols and conventions used in drawing flowcharts?**
   - **How do you represent decision points, loops, and function calls in a flowchart?**
5. **Translating Algorithms to Flowcharts:**
   - **How do you create a flowchart based on a given algorithm?**
   - **What considerations should be taken into account when translating complex algorithms into flowcharts?**
   - **Can you provide examples of flowcharts for common algorithms, such as bubble sort, binary search, or factorial calculation?**
6. **Analyzing Algorithms:**
   - **How do you analyze the time and space complexity of an algorithm?**
   - **What are Big O notation and how is it used to describe the performance of algorithms?**
   - **Can you compare the efficiency of different algorithms for solving the same problem?**
7. **Testing and Debugging Algorithms:**
   - **How do you test the correctness of an algorithm's implementation in C?**
   - **What strategies do you use to debug algorithms when they don't produce the expected results?**
   - **Can you provide examples of test cases and expected outcomes for validating algorithms?**

2. Variable Declaration

1. **Header Files and Preprocessor Directives:**
   - **What is the purpose of header files in C programming?**
   - **Can you explain the role of preprocessor directives like #include and #define?**
   - **How do you include standard library headers versus user-defined headers in a C program?**
2. **Main Function:**
   - **What is the significance of the main() function in a C program?**
   - **What are the valid return types and parameters of the main() function?**
   - **How do you pass command-line arguments to the main() function?**
3. **Function Declaration and Definition:**
   - **What is the syntax for declaring a function in C?**
   - **How do you define a function, and what are the components of a function definition?**
   - **Can you explain the concept of function prototypes and their importance?**
4. **Variable Declaration and Scope:**
   - **How do you declare variables in C, and what are the rules for naming variables?**
   - **What is variable scope, and how is it determined in C?**
   - **Can you explain the difference between local, global, and static variables?**
5. **Control Structures:**

- **What are the basic control structures in C programming?**
- **How do you use conditional statements like if, else if, and switch in C?**
- **Can you describe the syntax and use of loop structures such as for, while, and do-while?**

6. **Comments and Documentation:**
   - **What is the purpose of comments in a C program?**
   - **How do you write single-line and multi-line comments in C?**
   - **Can you explain the importance of documenting code for readability and maintainability?**
7. **Program Organization and Modularity:**
   - **How do you organize a C program into multiple source files?**
   - **What are header files, and how do they facilitate modular programming?**
   - **Can you describe the concept of separate compilation and linking in C?**

3. **Input / Output Statement**
   1. **Basic Input/Output Functions:**
      - **What are the basic input and output functions in C?**
      - **How do you use printf() and scanf() functions for formatted input and output?**
      - **Can you explain the difference between printf() and puts() functions?**
   2. **Formatted Input and Output:**
      - **How do you format output using format specifiers in printf()?**
      - **What are some commonly used format specifiers and their meanings?**
      - **How do you align output using width and precision specifiers?**
   3. **Reading and Writing Files:**
      - **How do you open, close, read, and write files in C?**
      - **What are the differences between text mode and binary mode file operations?**
      - **Can you explain the concept of file pointers and how they're used in file I/O?**
   4. **Error Handling in File I/O:**
      - **How do you check for errors during file I/O operations?**
      - **What are some common error codes returned by file I/O functions?**
      - **How do you handle file I/O errors gracefully in your C programs?**
   5. **Standard Streams:**
      - **What are standard input, output, and error streams in C?**
      - **How do you redirect input and output using <, >, and >> operators in the command line?**
      - **Can you explain the significance of stdin, stdout, and stderr streams in C programs?**
   6. **Buffered vs. Unbuffered I/O:**
      - **What is buffered I/O, and how does it improve performance?**
      - **How do you control buffering using setbuf() and setvbuf() functions?**
      - **When might you prefer unbuffered I/O over buffered I/O in C programs?**
   7. **Formatted Input using scanf():**
      - **How do you use scanf() function to read formatted input from the standard input?**
      - **What are some common issues and pitfalls associated with using scanf()?**
      - **How can you handle input validation and error checking when using scanf()?**

4. **Format Specifiers**

Understanding Format Specifiers:

What are format specifiers in C and why are they important?
Can you explain the role of format specifiers in input and output operations?
How do format specifiers help in formatting the display of data?

Basic Format Specifiers:

What is the format specifier for printing integers in decimal format?
How do you print characters and strings using format specifiers?

Can you explain the difference between `%d`, `%i`, `%c`, `%s`, and `%f` format specifiers?

Modifiers and Width Specifiers:

How do you use modifiers like `l`, `h`, `ll`, and `hh` with format specifiers?
What is the purpose of width specifiers like `width` and `precision` in format specifiers?
Can you provide examples of using width specifiers to control the display width of data?

Formatting Numeric Data:

How do you print floating-point numbers with specific precision using format specifiers?
What format specifier would you use to print integers in hexadecimal or octal format?
How can you align numeric data using format specifiers?

Formatting Date and Time:

Is there a specific format specifier for printing date and time in C?
How can you use format specifiers to format date and time values obtained from the system?
Can you provide examples of formatting date and time using `strftime()` function?

User Input and Validation:

How do you use format specifiers to read user input from the standard input (keyboard)?
What techniques can you use to validate user input using format specifiers?
Can you explain how to handle invalid input when using format specifiers for input operations?

Error Handling and Debugging:

How do you handle formatting errors when using format specifiers for input or output?
What debugging techniques can you use to identify issues related to format specifiers?
Can you provide examples of common pitfalls or mistakes when using format specifiers?

5. **Basic Syntax and Data Types:**
   - **What are the basic syntax rules of the C programming language?**
   - **Can you explain the differences between int, float, double, and char data types?**
   - **How do you declare and initialize variables in C?**
   - **What is the syntax for declaring a variable in C?**
   - **How do you initialize a variable during declaration?**
   - **Can you declare multiple variables of the same type in a single statement?**

   - **What are the basic data types available in C?**
   - **How do you determine the size of each data type on your system?**
   - **Can you explain the differences between signed and unsigned data types?**

   - **How do you declare a constant in C?**
   - **What is the significance of using the const keyword?**
   - **Can you define a constant with a value determined at runtime?**

   - **What are the arithmetic operators available in C?**
   - **How do you use relational and logical operators in C?**
   - **Can you explain the difference between prefix and postfix increment/decrement operators?**

   - **What are the different types of control structures in C?**
   - **How do you use conditional statements such as if, else if, and switch?**
   - **Can you demonstrate the use of loops like for, while, and do-while?**

- **How do you read input from the user in C?**
- **How do you display output to the console using printf?**
- **Can you explain the purpose of format specifiers in printf and scanf functions?**

- **What is type casting, and when is it necessary in C?**
- **How do you perform explicit type casting in C?**
- **Can you illustrate the potential pitfalls of implicit type conversion?**

- **What is the scope of a variable in C?**
- **How does the lifetime of a variable relate to its scope?**
- **Can you explain the concept of local and global variables in C?**

1. **Basic Data Types:**
    - **What are the basic data types available in C?**
    - **Can you explain the differences between integers, floating-point numbers, characters, and Booleans in C?**
    - **How do you declare variables of different data types in C?**
2. **Integer Data Types:**
    - **What is the range of values that can be stored in int, short, long, and long long data types?**
    - **How do you handle overflow and underflow when working with integer data types in C?**
    - **Can you explain the difference between signed and unsigned integers in C?**
3. **Floating-Point Data Types:**
    - **What is the difference between float and double data types in C?**
    - **How are floating-point numbers represented in memory?**
    - **What issues can arise when comparing floating-point numbers for equality in C?**
4. **Character Data Type:**
    - **How are characters represented in C, and what is the ASCII character encoding?**
    - **How do you declare character variables and initialize them with character literals?**
    - **Can you explain the difference between character literals and character constants in C?**
5. **Other Data Types:**
    - **What is the void data type used for in C?**
    - **How do you declare pointers and use them to store memory addresses?**
    - **Can you explain the purpose of typedef and enum in C, and how are they used?**
6. **Data Type Conversion:**
    - **What is implicit and explicit type conversion in C?**
    - **How do you perform type casting to convert between different data types?**
    - **What are the potential risks and issues associated with data type conversion in C?**
7. **User-Defined Data Types:**
    - **How do you define and use structures (struct) in C?**
    - **Can you give examples of using arrays and multidimensional arrays in C?**
    - **What are the advantages of using user-defined data types in C?**

6. **Control Structures:**
    - **Explain the usage of if-else statements in C with an example.**
    - **What are loops, and how do you use them in C? Provide examples of for, while, and do-while loops.**
    - **How do you use switch-case statements in C? Provide an example.**

1. **Conditional Statements (if, else if, else):**
- **What is the syntax of the if statement in C?**
- **How do you use the else if ladder to handle multiple conditions?**
- **Can you explain the difference between using nested if statements and else if statements?**
2. **Switch Statement:**

- **How does the switch statement differ from multiple nested if-else statements?**
- **What are the advantages of using a switch statement?**
- **When would you choose to use a switch statement over if-else statements?**

3. **Loops (for, while, do-while):**

- **Explain the syntax and usage of the for loop in C.**
- **What are the similarities and differences between the while and do-while loops?**
- **How do you break out of a loop prematurely using the break statement?**

4. **Loop Control Statements (break, continue):**

- **How does the break statement differ from the continue statement?**
- **When would you use the break statement inside a loop?**
- **Can you provide an example of using the continue statement to skip iteration in a loop?**

5. **Nested Control Structures:**

- **What are nested control structures, and why are they useful?**
- **Can you give an example of using nested loops in C?**
- **How do you avoid infinite loops when using nested control structures?**

6. **Error Handling:**

- **How can control structures be used for error handling in C?**
- **What are some common techniques for handling errors using if-else statements?**
- **Can you explain the concept of exception handling in C and how it differs from traditional error handling?**

7. **Control Structure Best Practices:**

- **What are some best practices for writing clear and maintainable control structures?**
- **How do you choose between different control structures based on the requirements of a problem?**
- **Can you identify any common mistakes or pitfalls to avoid when using control structures in C?**