Problem Solving with C Programming

Comprehensive Unit-wise Question Bank

Unit I: Introduction, Data Types, Operators, and Expressions

1 Mark Questions (Short Answer / Objective)

1. Who developed the C programming language?
2. What is the extension used to save a C program file?
3. Define a keyword in C.
4. Can a keyword be used as an identifier name in C?
5. State the rules for naming a valid identifier.
6. What is a constant in C?
7. Write the format specifier used for reading an unsigned integer.
8. Which function is used to display output on the screen in C?
9. What is the role of the preprocessor in C compilation?
10. Name the fundamental data types available in C.
11. What does the sizeof operator return?
12. Give an example of a type modifier in C.
13. Which operator is used to find the remainder of integer division?
14. What is the associativity of the assignment operator?
15. Differentiate between i++ and ++i.
16. Name the only ternary operator available in C.
17. What value is returned by a relational expression if the condition is false?
18. List the three bitwise operators in C.
19. What is the purpose of the scanf() function?
20. Why is C called a mid-level language?

2 Mark Questions

1. Differentiate between keywords and identifiers.
2. Explain the purpose of compilation in C programming.
3. What are data type qualifiers? Give examples.

4. Explain the difference between character constants and string constants.

5. What is the difference between / and % operators?

6. Write a short note on the conditional operator with syntax.

7. Explain precedence and associativity of operators with an example.

8. How are data types classified regarding memory representation and size?

9. What is type casting? Provide a brief example.

10. Describe compound assignment operators with a suitable code snippet.

11. Explain logical AND (&&) and logical OR (||) operators.

12. What is the role of stdio.h header file in C?

13. How do unary, binary, and ternary operators differ based on their operands?

14. Show how bitwise left shift and right shift operators affect a binary number.

15. Explain the importance of the header files and the #include directive.

5 Mark Questions

1. Describe the various stages involved in compiling and executing a C program.

2. Discuss the primitive data types supported by C along with their size and range on a 32-bit system.

3. Explain the different types of operators available in C with appropriate examples.

4. Elaborate on operator precedence and associativity rules in evaluating C expressions.

5. Write a C program to swap two variables using a third variable and without using a third variable.

6. Explain input and output operations in C covering formatted and unformatted operations.

7. What are constants? Explain the different types of constants used in C programming.

8. Discuss short-circuit evaluation in logical expressions with concrete examples.

9. Write a C program to calculate the area and perimeter of a circle where the radius is taken as user input.

10. Explain the concept of variables, their declaration, initialization, and memory mapping.

8 Mark Questions

1. Provide a comprehensive breakdown of the C execution pipeline from source code to executable file, explaining the role of the preprocessor, compiler, assembler, and linker.

2. Write a detailed essay on the operators in C, ranking them by precedence categories, and provide code examples showing how parentheses override default associativity.

3. Construct a complete C program that takes an integer from the user, determines if it is even or odd using the bitwise AND operator, and prints its size in bytes using the sizeof operator. Explain every line of code.

4. Discuss memory representation, modifiers (signed, unsigned, long, short), and limits of numerical data types in C, including how overflow occurs when values exceed ranges.

5. Create a C program to evaluate a complex mathematical expression containing arithmetic, relational, and logical operators, demonstrating step-by-step evaluation according to priority rules.

Unit II: Decision Control Structures, Loops, and Arrays

1 Mark Questions (Short Answer / Objective)

1. Write the syntax of a simple if statement.

2. What happens if the test condition in an if...else ladder matches nothing?

3. Which keyword handles unhandled cases in a switch statement?

4. What is the consequence of omitting a break statement in a switch case?

5. Which loop guarantees execution at least once?

6. Define an infinite loop.

7. What is the purpose of the continue statement?

8. Name the statement used to jump unconditionally out of a loop.

9. What is an array index or subscript?

10. What is the starting index of any array in C?

11. How is a two-dimensional array declaration structured?

12. Write the null character symbol that terminates a C string.

13. Name the header file required to work with string functions like strlen.

14. How do you find the total number of elements in a one-dimensional array using sizeof?

15. Can an array index be a negative number or a floating-point value?

16. What is a multi-dimensional array?

17. Name the statement that relies on label identifiers to alter control flow.

18. What is a character array?

19. Write down the limitation of standard fixed-size arrays in C.

20. How is a string initialized in a single line during declaration?

2 Mark Questions

1. Compare the if...else ladder with the switch statement.

2. Explain the execution mechanism of a for loop.

3. Differentiate between while and do...while loops.

4. Explain how a nested if...else structure works.

5. How does the break statement differ from the continue statement inside a loop?

6. Write a short snippet showing how to initialize a 1D array at the time of declaration.

7. What are the key limitations of arrays in C?

8. Show how elements of a 2D array are stored in computer memory sequentially.

9. Explain the use of the goto statement and why its usage is generally discouraged.

10. Describe how a string is represented as a character array in C memory.

11. Write a small program segment to read a string containing spaces using gets() or scanf().

12. Illustrate how to access an element located at the second row and third column of a 2D array.

13. What is the purpose of a loop counter variable? Give an example.

14. Explain what happens during array boundary violations or out-of-bound errors.

15. How do you copy one array into another array manually?

5 Mark Questions

1. Explain the working of a switch...case statement with syntax, rules, and a working code example.

2. Write a C program to find the largest of three numbers using nested if...else statements.

3. Describe the three loop structures available in C with their respective syntaxes and flowcharts.

4. Write a C program to check whether a given input number is a prime number or not.

5. Explain the concept of a 1D array, detailing its declaration, initialization, and how elements are accessed.

6. Write a C program to perform matrix addition using two-dimensional arrays.

7. Discuss character arrays and strings in C, explaining how strings are read, stored, and printed.

8. Explain the limitations of C arrays regarding size modifications and type uniformity.

9. Write a C program to find both the maximum and minimum numbers present within a user-defined 1D array.

10. Explain how nested loops operate, providing a code example that prints a star triangle pattern.

8 Mark Questions

1. Write a complete C program to perform matrix multiplication after validating the inner dimensions of both matrices using 2D arrays, and provide a clear dry run of the loop logic.

2. Compare entry-controlled and exit-controlled loops in detail. Write distinct programs using while and do...while to reverse a user-entered number and sum its digits.

3. Write a menu-driven C program using a switch statement to perform basic calculator functions (+, -, *, /) on two floating-point inputs inside an infinite loop that terminates only when a specific exit code is inputted.

4. Explain string handling without built-in library functions by writing a program that calculates string length, concatenates two strings, and reverses a string using standard loops.

5. Design an array-based C program to implement the bubble sort algorithm to arrange an array of integers in ascending order, accompanied by a step-by-step description of the sorting pass mechanics.

Unit III: Pointers, Functions, and Storage Classes

1 Mark Questions (Short Answer / Objective)

1. What is a pointer variable in C?
2. Which operator is known as the address-of operator?
3. Name the indirection or dereferencing operator.
4. What is a NULL pointer?
5. Define a wild pointer.
6. How does a dangling pointer occur?
7. What is a generic pointer in C?
8. Name the function used to allocate memory dynamically at runtime.
9. What does calloc() initialize allocated memory to?
10. Which function is used to deallocate dynamically allocated memory?
11. Define a function declaration or prototype.
12. What is a recursive function?
13. Which built-in function compares two strings alphabetically?
14. Name the default storage class for local variables.
15. Where are register storage class variables preferably stored?
16. What is the scope of a variable declared with the static storage class?
17. Which keyword is used to access a global variable declared in another file?
18. What string handling function appends one string to another?
19. Does pointer arithmetic depend on the data type of the pointer?
20. What is an array of pointers?

2 Mark Questions

1. Differentiate between an address-of operator (&) and an indirection operator (*).
2. Explain the concept of a wild pointer and how to avoid it.
3. What is a void or generic pointer? How do you dereference it?
4. Differentiate between malloc() and calloc() functions.
5. Explain the importance of the free() function in dynamic memory management.

6. What is the difference between a function prototype and a function definition?

7. Show how a string is accessed and traversed using a pointer.

8. Explain the difference between call by value and call by reference.

9. What is a dangling pointer? Provide a scenario where it is created.

10. Discuss the purpose of the realloc() function.

11. Explain the static keyword usage inside a function block.

12. What does strcmp() return when comparing two identical strings, and when comparing distinct ones?

13. Define pointer arithmetic rules regarding adding integers to pointers.

14. Explain what scope and lifetime mean for a variable.

15. How is an array name treated like a constant pointer in C?

5 Mark Questions

1. Explain the different types of pointers (NULL, wild, dangling, and generic) with short descriptive examples for each.

2. Differentiate between call by value and call by reference parameter passing mechanisms with distinct C program illustrations.

3. Explain dynamic memory allocation in C, highlighting the syntax and differences between malloc(), calloc(), realloc(), and free().

4. Write a recursive C function to calculate the factorial of a given positive integer and explain its call stack progression.

5. Discuss the four storage classes available in C (auto, register, static, extern) regarding their scope, lifetime, storage location, and initial values.

6. Write a C program that demonstrates how an array can be represented and processed entirely using pointer arithmetic.

7. Explain the following string handling functions with syntax and code snippets: strlen(), strcpy(), and strcat().

8. Write a function that accepts an array of integers and its size as arguments and returns the sum of the elements.

9. Explain the relationship between an array of pointers and multi-dimensional arrays with an architectural explanation.

10. Write a C program to find the occurrence of a substring within a main string using the strstr() function.

8 Mark Questions

1. Write a complete, modular C program to swap the values of two variables using pointers, and implement another custom function to find the length of a string using pointers without using library headers.

2. Explain the concept of recursion deeply. Write a recursive C program to generate the Fibonacci series up to N terms, and analyze its structural flow, tracking base and recursive cases.

3. Build an extensive demonstration program showcasing all four storage classes (auto, register, static, extern), utilizing multiple functions and source segments to illustrate differences in scope, lifetime, and initial state.

4. Design a dynamic memory management program that prompts the user for an initial array size, allocates memory via malloc(), populates it, expands it later via realloc(), sorts the data, and explicitly cleans memory with free().

5. Discuss pointer representations of arrays and arrays of pointers. Write a program that manages a list of variable-length strings using a pointer-to-pointer structure or pointer array to sort string arrays alphabetically.

Unit IV: Structures, Unions, File Management, and Command Line Arguments

1 Mark Questions (Short Answer / Objective)

1. What keyword is used to declare a structure in C?

2. What keyword is used to declare a union?

3. How much memory does a union allocate for its members?

4. Which operator is used to access structure members via a structure variable?

5. Which operator is used to access structure members via a pointer variable?

6. What is a nested structure?

7. Define a self-referential structure.

8. What is a bit-field in C?

9. Name the built-in structure type used to handle files in C.

10. Which function opens a file stream?

11. Write the file mode string used to open a file for appending data.

12. What does fopen() return if a file cannot be opened safely?

13. Which function is used to close an open file stream?

14. Name a function used to read a single character from a file.

15. What function writes a block of raw data or array elements into a binary file?

16. What does EOF stand for in file operations?

17. What are command line arguments in C?

18. What does the argc argument variable hold in a main() function?

19. What data type is argv[] in a command line argument setup?

20. Which function resets the file position indicator back to the beginning of a file?

2 Mark Questions

1. Define a structure and show its general initialization syntax.

2. Differentiate between structural definitions and structural variable instances.

3. What is the fundamental difference in memory allocation between a structure and a union?

4. Explain how an array of structures is declared and initialized.

5. What is a self-referential structure, and where is it primarily applied?

6. Show how a structure variable can be passed as an argument to a function.

7. Explain the purpose of file opening modes "r+" and "w+".

8. What is the significance of error handling in file input/output streams?

9. Explain the use of the feof() function in C file management.

10. What information does argv hold when executing a program with command line inputs?

11. Describe the utility of bit-fields inside structures.

12. Write a short snippet showing how to read structured records from a file using fscanf().

13. How do sequential access and random access files differ during operations?

14. Explain the usage of fseek() parameters for relative file positioning.

15. Describe how ftell() reports current positions inside a target file stream.

5 Mark Questions

1. Compare structures and unions in detail, explaining their memory layout diagrams and member accessibility rules.

2. Write a C program to store and display records of five students (including Roll No, Name, and Marks) using an array of structures.

3. Explain the concepts of nested structures and self-referential structures with clear code templates.

4. Write a C program to copy the entire text content of one source file into a destination file character by character.

5. Explain file opening modes ("r", "w", "a", "rb", "wb", "ab") along with their behaviors if files do or do not exist.

6. Discuss error handling during file operations, explaining functions like ferror() and perror().

7. Write a C program to read integer numbers from a file, separate even and odd numbers, and store them into two distinct output files.

8. Describe command line arguments in C, explaining the purpose and mechanics of argc and argv with a sample program.

9. Explain random access file management functions, detailing the parameters and actions of fseek(), ftell(), and rewind().

10. Write a C program that passes a structure pointer to a function to modify its members and print them out.

8 Mark Questions

1. Create a complete, functional file management C program that writes an array of employee structures containing fields for ID, Name, and Salary to a binary file using fwrite(), then reads them back via fread() and displays them on screen.

2. Discuss structures and functions comprehensively. Write a program that implements an inventory system handling parts records where structures are passed by value to a sorting function and passed by pointer reference to an updating function.

3. Write a C program that accepts a file name and a search keyword as command line arguments (argc/argv), scans the file for that keyword, counts its occurrences, and prints the total count with error trapping for missing parameters.

4. Detail the differences between sequential text file access and random access file mechanics. Write a tracking program using fseek(), ftell(), and structures to modify a specific record directly at an offset without rewriting the entire file.

5. Write an essay accompanied by a fully developed program explaining memory alignment, struct padding, and how the implementation of bit-fields enables compact layout designs for low-level register or packet tracking in C programming.