2 mark

1. Differentiate between system software and application software.
2. Write a C program to check if a number is even or odd using if-else statements.
3. Define the term "array" and write a C program to declare and initialize an integer array of size 5.
4. What is a loop? Write an example of a `for` loop in C.
5. Define computer hardware and software with examples.
6. Explain the different types of operators in C with examples.
7. Explain the role of the `return` statement in a C function.
8. Differentiate between global and local variables in C.
9. Define and give an example of conditional statements in C.
10. Differentiate between a compiler and an interpreter.
11. Differentiate between "call by value" and "call by reference" in C functions.
12. Write the difference between a flowchart and pseudocode.
13. What are the building blocks of algorithms?
14. Define the term "algorithm" and describe its characteristics.
15. Explain the difference between compilation and linking in the context of C programming.
16. Draw a flowchart for a C program calculating the area of a square.
17. What is a variable? Write a single line of C code to declare and initialize an integer variable with the value of 10.
18. Describe the structure of a C program with an example.
19. List and describe any three basic data types in C.
20. Write a C program using if-else statements.
21. Explain the purpose of the `#include` directive in a C program.
22. Explain the difference between `while` and `do-while` loops with examples.
23. Given a sample C program, identify two keywords and explain their functions briefly:
   ```c
   #include <stdio.h>
   int main() {
       int number = 10;
       printf("The value of number is: %d\n", number);
       return 0;
   }
   ```
24. What is a function? Write a sample C function to add two numbers.
25. Differentiate Iteration and Recursion.

13/14 marks

1. Discuss variables in C programming including their definition, purpose, declaration and initialization with examples, scope and lifetime, and best practices for naming.

2. Explain the steps involved in design and analysis during algorithmic problem solving. Also, compare an algorithm with pseudocode.

3. Explain the concepts of keywords, identifiers, and delimiters in the C programming language. Provide examples for each and discuss their significance in the structure and readability of C code.

4. Describe the basic structure of a C program, detailing each component, such as preprocessor directives, the main function, variable declarations, statements, and the return statement. Provide an example program to illustrate your explanation.

5. Write a C program to implement a simple calculator and explain each step of the program.

6. Examine the key characteristics of both iteration and recursion, discussing their strengths and weaknesses in algorithm development, as well as their impact on efficiency and complexity in problem-solving.

7. Compare and contrast pseudo code, flowcharts, and programming languages as notations for representing algorithms with an example.

8. Define what a token is in the context of the C programming language. Discuss its significance in structuring a C program.

9. Write the algorithm for finding the factorial of a number.

10. Discuss the different categories of programming languages and provide an example of each category.

11. Explain the building blocks of algorithms with examples of statements, state, control flow, and functions.

12. Draw the flowchart for finding the greatest among three numbers.

13. What are data types in C? List the primary data types and explain the purpose of each with examples.

14. Describe the differences between compiled and interpreted programming languages. Provide examples and discuss how each approach affects program performance and portability.

15. Explain the role of header files in C programming. How do they support modular programming, and why are standard libraries like `stdio.h` important?

16. Explain the concept of memory allocation in C with examples of static and dynamic memory. Describe how `malloc`, `calloc`, and `free` functions work in dynamic memory management and their significance in C programming.

17. Consider a C program that calculates the sum of an integer and a floating-point number. Discuss its significance in structuring a C program.

18. Discuss the significance of constants, variables, keywords, identifiers, and delimiters in the provided C program example.

19. What is the impact of algorithm efficiency on real-world applications? Provide examples where efficiency plays a crucial role.

20. Describe how debugging tools can assist in developing and optimizing C programs. Provide examples of commonly used debugging tools.