Long Questions

1. What is the difference between storing data in text format versus binary format?
2. How do you create a new file to store text and write a message into it using a high-level programming language?
3. Describe how to open a file and read its contents line by line.
4. What is the procedure for opening a binary file for reading and loading its content into a variable?
5. How can data be added to the end of an existing file without removing its current contents?
6. In what ways does appending data to a binary file differ from doing so in a text file?
7. How is a data structure written to a binary file using a programming language like C?
8. Explain how to read a data structure from a binary file and display its values.
9. How do you use a specific function to move the file pointer to a designated location within a file?
10. What method would you employ to determine the current position of the file pointer?
11. Describe the use of a function designed to reset the file pointer to the beginning of a file.
12. How can a particular record in a binary file be modified directly without having to rewrite the entire file?
13. Give an example of how to insert data into a specific position in a file using file pointer manipulation functions.
14. What strategies should be employed for handling errors that occur while performing file operations?
15. Can you describe a use case where it might be necessary to employ both file pointer resetting and positioning functions in handling a file, and explain the rationale?

16. How does a function in programming enhance code reusability and readability? Provide examples to illustrate your points.

17. Describe the differences between local and global variables in the context of functions with examples.

18. What is meant by the 'scope' of a variable, and how does it affect variable visibility and lifetime within a program?

19. How can functions return multiple values in C, considering it supports only single return values directly?

20. Illustrate with code how to pass an array to a function for modification. Discuss the implications for memory and efficiency.

21. Explain with examples the use of const keyword with pointers when passing an array to a function.

22. Describe the steps involved in passing a multidimensional array to a function. Provide a code snippet demonstrating this.

23. What is recursion, and how does it differ from iterative solutions in terms of execution flow and memory usage?

24. Provide an example of a recursive function that demonstrates the concept of a base case and recursive case.

25. Discuss the potential drawbacks of using recursion, including the risk of stack overflow and inefficiency. Provide examples.

26. How do dynamic memory allocation and pointer arithmetic enable the manipulation of arrays within functions? Include code examples.

27. Explain the use of static variables in recursive functions. How do they behave differently from non-static variables in such contexts?

28. Provide an example of a recursive algorithm for solving a common problem (other than factorial or Fibonacci) and discuss its time complexity.

29. Demonstrate the use of inline functions in C for optimizing small, frequently called functions. Compare its performance implications.

30. Discuss how variadic functions can be implemented in C to accept an arbitrary number of arguments. Provide an example with code.

31. Explain the process of dynamic memory allocation in C and its advantages over static memory allocation.

32. Describe how the malloc function is used to allocate memory dynamically. What precautions should be taken when using it?

33. What is the difference between malloc and calloc in terms of initialization of the allocated memory?

34. How does the realloc function work, and in what scenarios is it particularly useful?

35. Discuss the importance of freeing dynamically allocated memory and the consequences of failing to do so.

36. Provide an example of dynamically allocating memory for a single variable of type int and then freeing that memory.

37. Explain how to dynamically allocate memory for an array of integers. How does this process differ from allocating memory for a single integer?

38. What are the risks of memory leaks in C programs, and how can they be detected and prevented?

39. Describe how pointer arithmetic can be used to access and modify elements in a dynamically allocated array.

40. How can you dynamically allocate memory for a two-dimensional array using pointers? Explain the process step by step.

41. Discuss the role of the sizeof operator in dynamic memory allocation, providing examples of its use in allocating memory for different data types.

42. Explain the concept of memory fragmentation. How does dynamic memory allocation contribute to it, and what can be done to mitigate its effects?

43. What are the best practices for managing dynamically allocated memory in large programs to avoid memory leaks and undefined behavior?

44. Write a C program to dynamically allocate memory for an array of floats, input values from the user, and then free the allocated memory.

45. Demonstrate with a code example how to dynamically create a matrix (2D array) of int data type, assign values to it, and then free the memory.

46. How do you determine the nature of the roots of a quadratic equation before actually finding the roots?

47. Can a quadratic equation have one real and one imaginary root? Justify your answer.

48. How does the discriminant of a quadratic equation affect the roots? Provide examples.

49. Explain the process of finding roots of a quadratic equation when the coefficient of x^2 is greater than 1.

50. Discuss the limitations of the quadratic formula. Are there equations where it cannot be applied?

51. Describe an algorithm to find the minimum number in a set of integers. What is its time complexity?

52. Can the same algorithm used for finding the minimum in a set be used to find the maximum? Explain.

53. How would you modify your algorithm to handle a set containing both positive and negative numbers?

54. Discuss how finding the maximum number in a set would change if the set were sorted.

55. Illustrate with an example how you can find both the minimum and maximum numbers in a single pass through the set.

56. Describe an algorithm to check if a number is prime. What is the basic idea behind it?

57. How does the efficiency of your prime-checking algorithm change with the size of the input number?

58. Can your prime-checking algorithm be optimized for very large numbers? If so, how?

59. Discuss the limitations of trial division method in prime number checking.

60. How would you modify your algorithm to list all prime numbers below a given number N?

61. Compare linear search and binary search in terms of time complexity.

62. Why can't binary search be applied to an unsorted array?

63. Explain how binary search works. Why is it more efficient than linear search on sorted arrays?

64. Can binary search be used on a dataset where elements are sorted in descending order?

65. Discuss how the performance of linear search is affected as the size of the array increases.

66. Explain the basic principle of bubble sort and its time complexity.

67. Why is insertion sort more efficient than bubble sort in certain scenarios?

68. Describe selection sort and compare it with bubble sort in terms of number of swaps made.

69. Can bubble sort be considered efficient for large datasets? Why or why not?

70. How does the insertion sort behave on a nearly sorted array?

71. Explain the concept of Big O notation with an example.

72. How does understanding the time complexity of an algorithm help in real-world applications?

73. Compare the time complexity of linear search and binary search algorithms.

74. Why is it important to consider worst-case complexity when analyzing an algorithm?

75. Give an example of an algorithm with $O(n^2)$ complexity and explain why it's quadratic.