Answers for 100 Viva Questions
Data Structures and Applications (BCS304)

1. What is a data structure?

Method of organizing data in memory

2. Why do we need data structures?

To manage large data efficiently and perform operations like insertion, deletion, searching, and sorting easily.

3. Classify data structures with examples.

Primitive (int, float) and Non-primitive (arrays, linked lists, stacks, trees).

4. Difference between primitive and non-primitive data structures?

Primitive stores single values; non-primitive stores multiple values.

5. What is a linear data structure?

Data elements are arranged sequentially. Example: array, stack.

6. What is a non-linear data structure?

Data elements are arranged hierarchically. Example: tree, graph.

7. Define stack.

Stack is a linear data structure that follows LIFO (Last In First Out).

8. Basic operations of a stack?

Push, Pop, Display.

9. What is stack overflow?

Condition when push operation is attempted on a full stack.

10. What is stack underflow?

Condition when pop operation is attempted on an empty stack.

11.Queue is a linear data structure that follows FIFO (First In First Out).

12. Difference between stack and queue?

Stack uses LIFO; Queue uses FIFO.

13. What is a string?

A sequence of characters terminated by a null character.

14. What is pattern matching?

Process of finding a pattern string within a text string.

15. Purpose of string pattern matching?

Used to search text efficiently.

16. What is postfix expression?

Operator comes after operands. Example: AB+.

17. What is infix expression?

Operator comes between operands. Example: A+B.

18. Why postfix evaluation is easier?

No need for operator precedence or parentheses.

19. What is dynamic memory allocation?

Allocating memory at runtime.

20. Name dynamic memory allocation

malloc(), calloc(), realloc(), free().

21. What is a linked list?

A collection of nodes where each node contains data and link field which stores address of other node.

22. Why linked list is preferred over arrays?

Dynamic size and efficient insertion and deletion.

23. What is a node?

A basic unit of a linked list containing data and link.

24. Define singly linked list.

Each node points to the next node only.

25. Define doubly linked list.

Each node points to both previous and next nodes.

26. Define circular linked list.

Last node points back to the first node.

27. What is header node?

A special node used to store list information.

28. Advantages of linked list?

Dynamic size, no memory wastage, easy insertion.

29. What is insertion?

Adding a new node to the linked list.

30. What is deletion?

Removing a node from the linked list.

31. What is linear queue?

Queue implemented using array in linear fashion.

32. What is circular queue?

Queue where last position connects to first position.

33. Why circular queue is better?

Avoids memory wastage.

34. What is queue overflow?

Insertion when queue is full.

35. What is queue underflow?

Deletion when queue is empty.

36. Polynomial representation using linked list?

Each node represents a term with coefficient and exponent.

37. Why linked list for polynomial operations?

Dynamic size and easy operations.

38. What is multiple queue?

More than one queue in same memory.

39. What is priority queue?

Queue where elements are processed based on priority.

40. Application of queue?

CPU scheduling, printer queue.

41. What is a tree?

Non-linear hierarchical data structure.

42. What is a binary tree?

Each node has at most two children.

43. What is a leaf node?

Node with no children.

44. Degree of a node?

Number of children of a node.

45. What is tree traversal?

Visiting all nodes of a tree.

46. Inorder traversal?

Left, Root, Right.

47. Preorder traversal?

Root, Left, Right.

48. Postorder traversal?

Left, Right, Root.

49. Level order traversal?

Traversal level by level.

50. Threaded binary tree?

Binary tree with threads replacing NULL pointers.

51. In-threaded binary tree?

Right NULL pointers point to inorder successor.

52. What is sparse matrix?

Matrix with many zero elements.

53. Why sparse matrix representation?

To save memory.

54. Sparse matrix using linked list?

Stores only non-zero elements.

55. What is forest?

Collection of disjoint trees.

56. Forest to binary tree conversion?

Using left-child right-sibling representation.

57. What is selection tree?

Tree used to select minimum or maximum element.

58. What is leftist tree?

Heap where left subtree is heavier.

59. Advantage of leftist tree?

Efficient merging.

60. Optimal binary search tree?

BST with minimum search cost.

61. What is a graph?

Set of vertices and edges.

62. Vertex and edge?

Vertex is a node, edge is a connection.

63. Directed graph?

Edges have direction.

64. Undirected graph?

Edges have no direction.

65. Adjacency matrix?

2D array representing edges.

66. Adjacency list?

List representation of edges.

67. Difference between matrix and list?

Matrix uses more space; list is space-efficient.

68. Graph traversal?

Visiting all vertices of a graph.

69. DFS?

Depth First Search goes deep first.

70. BFS?

Breadth First Search goes level by level.

71. Data structure in DFS?

Stack or recursion.

72. Data structure in BFS?

Queue.

73. DFS vs BFS?

DFS goes deep; BFS goes wide.

74. Connected graph?

Path exists between all vertices.

75. Disconnected graph?

Some vertices are not connected.

76. Self-loop?

Edge from a node to itself.

77. Parallel edges?

Multiple edges between same nodes.

78. Weighted graph?

Edges have weights.

79. Elementary graph operations?

Insert, delete, traverse.

80. Application of graph traversal?

Routing, social networks.

81. What is hashing?

Technique to map keys to addresses.

82. Hash table?

Array used to store keys.

83. Hash function?

Function that computes(produces) hash value

84. Remainder method?

H(K) = K mod m.

85. Collision?

Two keys map to same address.

86. Why collision occurs?

Limited table size.

87. Linear probing?

Sequentially checking next locations.

88. Linear probing formula?

(h(k) + i) % m.

89. Chaining?

Linked list at each hash index.

90. Chaining vs linear probing?

Chaining uses lists; probing uses array.

91. Static hashing?

Fixed table size.

92. Dynamic hashing?

Table size changes dynamically.

93. Load factor?

Ratio of number of keys to table size.

94. Primary clustering?

Grouping of keys due to probing.

95. Rehashing?

Recomputing hash table.

96. AVL tree?

Self-balancing binary search tree.

97. Why AVL is balanced?

Height difference is at most one.

98. Rotation in AVL?

Operation to balance tree.

99. Red-Black tree?

Self-balancing BST using color properties.

100. Real-time application of hashing?

Databases and password storage.