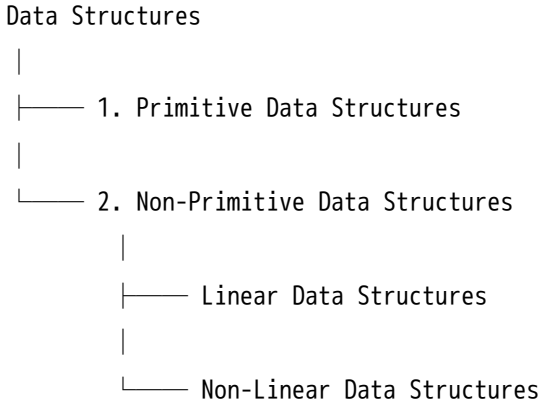
Data Structure Unit 1 — 7 Mark Answers

1. Explain Classification of Data Structures

Data Structure means a way of storing and organizing data in memory.

Data structures are mainly classified into two types:



1. Primitive Data Structures

These are basic data types provided by programming languages.

Examples:

```
int, float, char, double
```

2. Non-Primitive Data Structures

These are advanced data structures made using primitive data types.

Examples:

```
Array, Linked List, Stack, Queue, Tree, Graph
```

Linear Data Structures

In linear data structure, data is arranged in sequence.

Examples:

```
Array, Linked List, Stack, Queue
```

Non-Linear Data Structures

In non-linear data structure, data is not arranged sequentially. It may be hierarchical or network-based.

Examples:

```
Tree, Graph
```

Conclusion

Classification helps us choose the correct data structure according to the problem.

---

2. Explain ADT with Example

ADT stands for **Abstract Data Type**.

ADT defines **what operations can be performed**, but does not explain **how they are implemented**.

Simple Definition

ADT is a logical model of data and operations.

Example: Stack ADT

A stack follows **LIFO** principle:

```
Last In First Out
```

Stack Operations

```
push()   → insert element
pop()    → delete top element
peek()   → view top element
isEmpty() → check stack empty or not
```

Important Point

Stack can be implemented using:

```
Array
Linked List
```

But ADT only tells operations, not implementation.

Advantages of ADT

- Hides implementation details
- Makes program simple

- Improves reusability
- Easy to modify

Conclusion

ADT helps programmers focus on operations instead of internal coding details.

---

3. Explain Operations on Data Structures

Data structure operations are actions performed on stored data.

1. Traversing

Visiting each element one by one.

Example:

```
Printing all array elements
```

2. Insertion

Adding a new element into data structure.

Example:

```
Adding 50 in an array
```

3. Deletion

Removing an element from data structure.

Example:

```
Deleting a node from linked list
```

4. Searching

Finding a particular element.

Example:

```
Searching roll number 25
```

5. Sorting

Arranging elements in order.

Example:

```
10, 30, 20 → 10, 20, 30
```

6. Merging

Combining two data structures.

Example:

```
Array A + Array B = New Array C
```

Cost Estimation

Every operation has cost:

```
Time Complexity → Time taken
Space Complexity → Memory used
```

Conclusion

Operations help in managing data efficiently inside memory.

---

4. Explain Linked List with Memory Representation

A **Linked List** is a linear data structure where elements are stored in nodes.

Each node contains:

```
1. Data
2. Address of next node
```

Node Structure in C

```
struct node
{
    int data;
    struct node *next;
};
```

Simple Linked List

```
10 → 20 → 30 → NULL
```

Memory Representation

Unlike arrays, linked list nodes are not stored continuously.

```
Address     Data     Next

1000        10       2500
2500        20       1800
1800        30       NULL
```

Here:

- First node stores data 10
- Its next pointer stores address 2500
- Last node points to NULL

Advantages

- Dynamic size
- Easy insertion
- Easy deletion
- Memory used as needed

Disadvantages

- Extra memory required for pointer
- Direct access is not possible
- Searching is slower than array

Conclusion

Linked list is useful when size of data changes frequently.

---

5. Difference Between Array and Linked List

Array	Linked List
Fixed size	Dynamic size
Stored in continuous memory	Stored in non-continuous memory
Direct access possible	Direct access not possible
Insertion and deletion difficult	Insertion and deletion easy
No extra pointer memory	Extra pointer memory required
Memory may be wasted	Memory used as needed

Array Example

```
10 20 30 40
```

Stored continuously.

Linked List Example

```
10 → 20 → 30 → NULL
```

Stored using pointers.

Conclusion

Array is best when size is fixed. Linked list is best when frequent insertion and deletion are required.

DS Unit 1: Important 14-Mark Answers

1. Explain Arrays and Linked Lists in Detail

Array

An **array** is a linear data structure that stores elements of the same data type in continuous memory locations.

Example:

```
int arr[5] = {10, 20, 30, 40, 50};
```

Memory representation:

```
Index:    0    1    2    3    4
Value:   10   20   30   40   50
```

Features of Array

Array stores similar type of data. Its size is fixed. Elements are accessed using index number. The first element is stored at index 0.

Advantages of Array

Arrays provide fast access because any element can be accessed directly using index. They are simple to use and easy to traverse.

Disadvantages of Array

The size of array is fixed. Insertion and deletion are difficult because shifting is required. Memory may be wasted if array is not fully used.

Linked List

A **linked list** is a linear data structure in which elements are stored in nodes. Each node contains data and address of the next node.

Node structure:

```
struct node {
    int data;
    struct node *next;
};
```

Representation:

```
10 → 20 → 30 → NULL
```

Features of Linked List

Linked list uses dynamic memory allocation. Nodes are not stored continuously. Each node is connected using pointer.

Advantages of Linked List

Linked list size can grow or shrink during runtime. Insertion and deletion are easy. Memory is used as required.

Disadvantages of Linked List

Extra memory is required for pointer. Direct access is not possible. Searching is slower compared to array.

Difference

Array	Linked List
Fixed size	Dynamic size
Continuous memory	Non-continuous memory
Direct access possible	Direct access not possible
Insertion/deletion difficult	Insertion/deletion easy
No pointer required	Pointer required

Conclusion

Array is useful when size is fixed and fast access is needed. Linked list is useful when frequent insertion and deletion are required.

---

2. Explain Types of Linked List with Diagrams

A linked list is a collection of nodes connected using pointers.

1. Singly Linked List

In singly linked list, each node contains data and address of next node.

Diagram:

```
START → [10 | next] → [20 | next] → [30 | NULL]
```

Advantages

It is simple to implement. It uses dynamic memory. Insertion and deletion are easy.

Disadvantages

Traversal is possible only in one direction. Extra memory is required for pointer.

---

2. Doubly Linked List

In doubly linked list, each node contains three parts:

```
Previous pointer | Data | Next pointer
```

Diagram:

```
NULL ← [prev | 10 | next] ⇄ [prev | 20 | next] ⇄ [prev | 30 | next] → NULL
```

Advantages

Traversal is possible in both directions. Deletion is easier compared to singly linked list.

Disadvantages

It requires more memory because two pointers are used.

---

3. Circular Linked List

In circular linked list, last node points back to the first node.

Diagram:

```
[10] → [20] → [30]
 ↑              ↓
```

Advantages

It is useful in circular operations such as CPU scheduling. No NULL pointer is required at the end.

Disadvantages

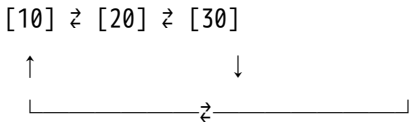
If not handled carefully, it may cause infinite loop.

---

4. Doubly Circular Linked List

It is a combination of doubly linked list and circular linked list.

Diagram:



Applications of Linked List

Linked lists are used in dynamic memory allocation, browser history, music playlists, stack, queue, polynomial manipulation and operating systems.

Conclusion

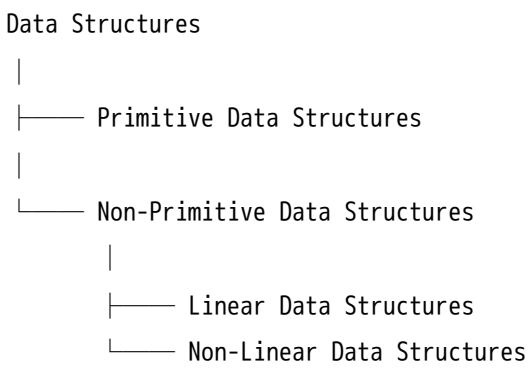
Different types of linked lists are used according to the requirement of traversal, insertion and deletion.

---

3. Explain Data Structure Classification with Examples

Data structure means organizing data in memory so that it can be used efficiently.

Classification:



1. Primitive Data Structures

These are basic data types provided by programming languages.

Examples:

```
int, float, char, double
```

Example:

```
int age = 20;
char grade = 'A';
```

2. Non-Primitive Data Structures

These are advanced structures created using primitive data types.

Examples:

```
Array, Linked List, Stack, Queue, Tree, Graph
```

Linear Data Structures

In linear data structure, elements are arranged sequentially.

Examples:

```
Array, Linked List, Stack, Queue
```

Array

Stores elements in continuous memory.

```
10 20 30 40
```

Linked List

Stores elements using nodes and pointers.

```
10 → 20 → 30 → NULL
```

Stack

Follows LIFO principle.

```
Last In First Out
```

Queue

Follows FIFO principle.

```
First In First Out
```

Non-Linear Data Structures

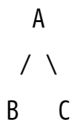
In non-linear data structure, elements are not arranged sequentially.

Examples:

```
Tree, Graph
```

Tree

Hierarchical structure.



Graph

Collection of vertices and edges.

```
A ---- B
|      |
C ---- D
```

Conclusion

Classification helps in selecting the best data structure for solving a particular problem.

---

4. Explain Memory Representation and Operations on Data Structures

Memory Representation

Memory representation means how data is stored inside computer memory.

Data structures can be stored in two ways:

1. Contiguous Memory Allocation

Memory locations are continuous.

Example: Array

```
Address: 1000  1004  1008  1012
Value:    10    20    30    40
```

Advantages

Fast access is possible. Index-based access is easy.

Disadvantages

Fixed size. Insertion and deletion are costly.

---

2. Linked Memory Allocation

Memory locations are not continuous. Each node stores address of next node.

Example: Linked List

```
Address   Data   Next
1000      10     2500
2500      20     1800
1800      30     NULL
```

Advantages

Dynamic memory allocation. Easy insertion and deletion.

Disadvantages

Extra pointer memory is required. Direct access is not possible.

Operations on Data Structures

1. Traversing

Visiting each element one by one.

Example:

```
Print all elements of array.
```

2. Insertion

Adding a new element.

Example:

```
Insert 25 between 20 and 30.
```

3. Deletion

Removing an element.

Example:

```
Delete 20 from linked list.
```

4. Searching

Finding a required element.

Example:

```
Search 50 in array.
```

5. Sorting

Arranging elements in order.

Example:

```
30, 10, 20 → 10, 20, 30
```

6. Merging

Combining two data structures.

Example:

```
Array A + Array B = Array C
```

Cost Estimation

Cost means time and memory required.

Time Complexity

Time required to perform operation.

Examples:

```
O(1), O(n), O(log n)
```

Space Complexity

Memory required to perform operation.

Conclusion

Memory representation and operations are important for analyzing efficiency of data structures.

---

5. Explain Polynomial Manipulation Using Linked List

Polynomial

A polynomial is an algebraic expression containing terms with coefficient and power.

Example:

```
5x² + 3x + 2
```

Here:

Term	Coefficient	Power
$5x^2$	5	2
$3x$	3	1
2	2	0

Why Use Linked List for Polynomial?

In polynomial, number of terms may change. Linked list is useful because it supports dynamic memory allocation.

Node Structure

Each polynomial term can be stored in one node.

```
struct poly {
    int coeff;
    int power;
    struct poly *next;
};
```

Representation

Polynomial:

```
5x² + 3x + 2
```

Linked list representation:

```
[5 | 2 | next] → [3 | 1 | next] → [2 | 0 | NULL]
```

Here:

```
First part = coefficient
Second part = power
Third part = next pointer
```

Polynomial Addition

Suppose:

```
P1 = 5x² + 3x + 2
P2 = 4x² + 2x + 1
```

Add same powers:

$5x^2 + 4x^2 = 9x^2$

$3x + 2x = 5x$

$2 + 1 = 3$

Result:

$9x^2 + 5x + 3$

Steps for Polynomial Addition Using Linked List

1. Create two linked lists for two polynomials.
2. Compare powers of both nodes.
3. If powers are same, add coefficients.
4. If power of first polynomial is greater, copy first node.
5. If power of second polynomial is greater, copy second node.
6. Move to next node.
7. Continue until both lists become empty.

Algorithm

```
Step 1: Start
Step 2: Read polynomial P1 and P2
Step 3: Compare power of current nodes
Step 4: If powers are equal, add coefficients
Step 5: If power of P1 is greater, copy P1 term
Step 6: If power of P2 is greater, copy P2 term
Step 7: Move pointer forward
Step 8: Display result polynomial
Step 9: Stop
```

Advantages

- Dynamic memory use
- Easy insertion of terms
- Efficient for sparse polynomials
- No memory wastage

Applications

- Algebraic computation
- Scientific calculators
- Computer algebra systems
- Engineering mathematics

Conclusion

Linked list is very useful for polynomial manipulation because each term can be stored dynamically and operations like addition and subtraction become easier.