Day 12 - 27th June 2025

**Task 1:**

**What do you understand about data structures?**

A data structure is a specialized format for organizing, storing, and managing data in a computer so that it can be accessed and modified efficiently. It's like a container that stores data in a specific way to serve particular purposes.

Data structures help in:

* Managing large amounts of data
* Making data manipulation more efficient
* Organizing data in a way that suits specific operations
* Improving program performance and efficiency

**Task 2:**

**What are the types of data structures**

Data structures can be broadly classified into two categories:

Primitive Data Structures:

1. Integer
2. Float
3. Character
4. Boolean
5. Double
6. String

Non-Primitive Data Structures:

A. Linear Data Structures:

1. Arrays
2. Linked Lists
3. Stacks
4. Queues

B. Non-Linear Data Structures:

1. Trees
2. Graphs
3. Hash Tables

C. Dynamic Data Structures:

1. Dynamic Arrays
2. Dynamic Lists

Each type has specific characteristics and use cases:

* Arrays: Fixed-size sequential collection
* Linked Lists: Sequential elements connected via pointers
* Stacks: LIFO (Last In First Out) structure
* Queues: FIFO (First In First Out) structure
* Trees: Hierarchical structure with parent-child relationships
* Graphs: Network structure with nodes and edges
* Hash Tables: Key-value pair structure for fast data retrieval

The choice of data structure depends on:

Type of data to be stored is decided based on:

1. Cost of operations
2. Memory usage
3. Ease of implementation

**Task 3:**

**What all operations can we do in Data structures?**

In Data Structures, we can perform several fundamental operations. Here are the main operations:

Insertion Operations:

* Adding elements to the data structure
* Can be done at beginning, end, or specific position
* Example: insert(element), push(), append()

Deletion Operations:

* Removing elements from the data structure
* Can delete from beginning, end, or specific position
* Example: delete(element), pop(), remove()

Traversal Operations:

* Visiting each element in the data structure
* Processing elements one by one
* Example: iterate through array, traverse linked list

Searching Operations:

* Finding specific elements in the data structure
* Linear search, binary search, etc.
* Example: search(key), find(element)

Sorting Operations:

* Arranging elements in a specific order
* Ascending or descending order
* Example: bubble sort, quick sort, merge sort

Updating Operations:

* Modifying existing elements
* Changing values at specific positions
* Example: update(position, newValue)

Merging Operations:

* Combining two data structures
* Creating a new structure from existing ones
* Example: merge two sorted arrays

Splitting Operations:

* Dividing data structure into smaller parts
* Creating sub-structures
* Example: splitting an array into two

Access Operations:

* Retrieving elements from specific positions
* Reading values without modification
* Example: get(index), peek()
* These operations vary in efficiency (time complexity) depending on:

Type of data structure used is decided based on:

Size of data

Implementation method

Position of element being operated on

Understanding these operations helps in:

Choosing right data structure for specific needs:

Optimizing program performance

Efficient data management

Problem-solving in programming

**Task 4:**

**What are static and dynamic arrays? Explain or summarize key points in a table like**

**Size, performance, memory, flexibility, limitations**

| **Aspect** | **Static Arrays** | **Dynamic Arrays** |
| --- | --- | --- |
| Size | • Fixed size at declaration  • Cannot be changed during runtime  • Size must be known at compile time | • Variable size  • Can grow or shrink during runtime  • Automatically resizes as needed |
| Performance | • Faster access time (O(1))  • No resizing overhead  • Better cache utilization  • Direct memory access | • Slightly slower due to resizing operations  • Amortized time complexity for insertion  • Additional overhead for size management |
| Memory | • Fixed memory allocation  • More memory efficient  • No extra space overhead | • Additional memory overhead  • Usually allocates more space than needed  • Typically doubles size when full |
| Flexibility | • Less flexible  • Cannot accommodate more elements than initial size  • Better for fixed-size data | • Highly flexible  • Can adapt to varying data sizes  • Good for unknown or changing data sizes |
| Limitations | • Risk of overflow  • Wastage of space if underutilized  • Cannot shrink to save memory | • Memory fragmentation possible  • Performance cost during resizing  • More complex memory management |

**8-bit representation of capital 'A':**

1. First, find ASCII value of 'A' = 65

2. Convert 65 to binary:

65 ÷ 2 = 32 (Remainder: 1)

32 ÷ 2 = 16 (Remainder: 0)

16 ÷ 2 = 8 (Remainder: 0)

8 ÷ 2 = 4 (Remainder: 0)

4 ÷ 2 = 2 (Remainder: 0)

2 ÷ 2 = 1 (Remainder: 0)

1 ÷ 2 = 0 (Remainder: 1)

Reading from bottom to top: 1000001

3. For 8-bit representation, add leading zeros:

01000001

Therefore, capital 'A' in 8-bit binary = 01000001

Quick check:

- 7th bit (64) = 1

- 0th bit (1) = 1

- All other bits = 0

- Total = 64 + 1 = 65 (ASCII value of 'A')

**Task 5:**

**What is the binary value of a?**

Hint ascii value is 97

Step 1: Convert 97 to binary

97 ÷ 2 = 48 (Remainder: 1)

48 ÷ 2 = 24 (Remainder: 0)

24 ÷ 2 = 12 (Remainder: 0)

12 ÷ 2 = 6 (Remainder: 0)

6 ÷ 2 = 3 (Remainder: 0)

3 ÷ 2 = 1 (Remainder: 1)

1 ÷ 2 = 0 (Remainder: 1)

Reading from top to bottom

8-bit representation of 'a' = 01100001

**Task 6:**

**Types of Computer memory with examples.. Explain ..**

1. Primary Memory (Main Memory)

• RAM (Random Access Memory)

Volatile memory

Examples: DDR3, DDR4, DDR5

Used for temporary data storage

• ROM (Read Only Memory)

Non-volatile memory

Examples: PROM, EPROM, EEPROM

Stores permanent instructions

1. Secondary Memory

• Hard Disk Drives (HDD)

Mechanical storage

Large capacity, slower access

• Solid State Drives (SSD)

Flash-based storage

Faster than HDD, more expensive

• Optical Drives

CD-ROM, DVD, Blu-ray

Removable storage media

1. Cache Memory

• L1 Cache

Smallest, fastest

Located closest to CPU

• L2 Cache

Larger than L1, slightly slower

• L3 Cache

Largest cache, shared between cores

Registers

1. CPU Registers

Fastest memory

Very small capacity

Used for immediate processing

1. Flash Memory

• USB drives

• Memory cards

• NAND flash storage

Key Characteristics:

• Speed: Registers > Cache > RAM > SSD > HDD

• Cost: Higher speed = Higher cost

• Capacity: Higher speed = Lower capacity

• Volatility: RAM is volatile, ROM/Secondary memory is non-volatile

Data Structures

Data structures in Java.docx in docs to study – for reference..