ASSIGNMENT 1

DATA STRUCTURE CLASSIFICATION, APPLICATIONS AND SYSTEM USAGE

Programming Language Used: C

1️⃣ Data Structure Classification

Data structures are ways of organizing and storing data so that it can be accessed and modified efficiently.

They are classified into:

Primitive Data Structures

Non-Primitive Data Structures

Linear Data Structures

Non-Linear Data Structures

2️⃣ Primitive Data Structures

Primitive data structures are basic built-in data types provided by a programming language. They store single values.

Examples:

Int

Float

Char

Double

Example in C:

```
#include <stdio.h>


Int main() {
  Int age = 21;
  Float height = 5.6;
  Char grade = 'A';
```

```
   Printf("Age: %d\n", age);
   Printf("Height: %.1f\n", height);
   Printf("Grade: %c\n", grade);


   Return 0;
}
```

Where They Are Applied

Storing student marks

Storing bank balances

Storing product prices

Counting items in a system

Why They Are Used

Simple and fast

Use less memory

Directly supported by hardware

Essential for building complex data structures

3️⃣ Non-Primitive Data Structures

Non-primitive data structures store multiple values. They are divided into:

Linear Data Structures

Non-Linear Data Structures

A. Linear Data Structures

In linear data structures, elements are arranged sequentially (one after another).

1. Array

An array is a collection of elements stored in contiguous memory locations. Each element is accessed using an index.

Example in C:

```
#include <stdio.h>


Int main() {
   Int numbers[5] = {10, 20, 30, 40, 50};


   For(int i = 0; i < 5; i++) {
      Printf("%d ", numbers[i]);
   }


   Return 0;
}
```

Applications of Arrays

Storing marks of students

Storing daily temperatures

Image processing (pixel storage)

Matrix operations in mathematics

Algorithms Used With Arrays

Linear Search

Binary Search

Bubble Sort

Quick Sort

Why Arrays Are Used

Fast access using index (O(1) time complexity)

Simple to implement

Efficient when size is fixed

2. Linked List

A linked list is a linear data structure where elements (nodes) are connected using pointers. Each node contains:

Data

Address of the next node

Example in C:

```
#include <stdio.h>

#include <stdlib.h>


Struct Node {

   Int data;

   Struct Node* next;

};


Int main() {

   Struct Node* head = (struct Node*)malloc(sizeof(struct Node));

   Head->data = 10;

   Head->next = NULL;


   Printf("Node data: %d", head->data);

   Return 0;

}
```

Applications

Music playlists

Browser history

Undo/Redo operations

Memory management

Why Linked Lists Are Used

Dynamic size (can grow/shrink)

Easy insertion and deletion

No need for contiguous memory

3. Stack

A stack follows the LIFO (Last In First Out) principle.

Operations:

Push (insert)

Pop (remove)

Example in C:

```
#include <stdio.h>
#define MAX 5

Int stack[MAX];
Int top = -1;

Void push(int value) {
  If(top == MAX-1)
    Printf("Stack Overflow\n");
  Else
    Stack[++top] = value;
}

Void pop() {
  If(top == -1)
```

```
        Printf("Stack Underflow\n");
    Else
        Top--;
}


Int main() {
    Push(10);
    Push(20);
    Pop();
    Return 0;
}
```

Applications

Function calls (call stack)

Expression evaluation

Undo feature

Browser back button

Algorithms Used

Depth First Search (DFS)

Expression conversion (Infix to Postfix)

Why Stack Is Used

Efficient for recursive operations

Easy implementation

Useful for reversing data

4. Queue

A queue follows the FIFO (First In First Out) principle.

Operations:

Enqueue (insert)

Dequeue (remove)

Example in C:

```
#include <stdio.h>
#define MAX 5

Int queue[MAX];
Int front = -1, rear = -1;

Void enqueue(int value) {
  If(rear == MAX-1)
    Printf("Queue Full\n");
  Else {
    If(front == -1) front = 0;
    Queue[++rear] = value;
  }
}

Int main() {
  Enqueue(10);
  Enqueue(20);
  Return 0;
}
```

Applications

CPU scheduling

Printing queue

Customer service systems

Network packet management

Algorithms Used

Breadth First Search (BFS)

Scheduling algorithms

Why Queue Is Used

Maintains order of execution

Fair resource allocation

Efficient process management

B. Non-Linear Data Structures

Non-linear data structures do not store data sequentially.

1. Tree

A tree is a hierarchical structure with:

Root node

Parent and child nodes

Example in C;

```
#include <stdio.h>

#include <stdlib.h>


Struct Node {

  Int data;

  Struct Node* left;

  Struct Node* right;

};


Int main() {
  Struct Node* root = (struct Node*)malloc(sizeof(struct Node));
  Root->data = 1;
  Root->left = NULL;
  Root->right = NULL;


  Printf("Root: %d", root->data);
  Return 0;
}
```

Applications

File systems

Database indexing (B-Trees)

HTML DOM

Artificial Intelligence decision trees

Algorithms Used

Tree Traversal (Inorder, Preorder, Postorder)

Binary Search Tree operations

Heap algorithms

Why Trees Are Used

Efficient searching

Hierarchical representation

Faster data retrieval

2. Graph

A graph consists of:

Vertices (nodes)

Edges (connections)

Example in C (Adjacency Matrix):

```
#include <stdio.h>


Int main() {
   Int graph[3][3] = {
      {0,1,1},
      {1,0,0},
      {1,0,0}
   };


   Printf("Graph created successfully");
   Return 0;
}
```

Applications

Social networks

GPS navigation

Network routing

Airline routes

Algorithms Used

Dijkstra's Algorithm

BFS

DFS

Minimum Spanning Tree

Why Graphs Are Used

Represent relationships

Solve shortest path problems

Model real-world networks

4️⃣ How Data Structures and Algorithms Work Within Systems

Data structures organize data in memory.

Algorithms define steps to process that data efficiently.

Together, they:

Improve speed

Reduce memory usage

Enhance system performance

Example of Systems

Banking System

Data Structure: Arrays / Trees

Algorithm: Searching and Sorting

Reason: Fast account lookup

GPS Navigation

Data Structure: Graph

Algorithm: Dijkstra’s algorithm

Reason: Finds shortest path

Operating System

Data Structure: Queue

Algorithm: Scheduling algorithm

Reason: Manage processes efficiently

Web Browser

Data Structure: Stack

Algorithm: Backtracking

Reason: Track visited pages