

Notes n Hindi

SUBJECT

Data Structure

PREPARED BY

TEAM NOTESINHINDI

<https://notesinhindi.com/>

2026 Edition

Unit 1: Basics of Data Structure

Classification of Data Structure, Pointers, Structure, aur Operations on Data Structures

Introduction

Data Structure एक ऐसा तरीका है जिससे data को computer की memory में इस तरह organize aur store किया जाता है कि उसे efficiently access aur modify किया जा सके। आसान भाषा में कहें तो, अगर data को systematic तरीके से arrange न किया जाए, तो उसे ढूँढना aur manage करना बहुत मुश्किल हो जाता है — इसीलिए Data Structures का concept इस्तेमाल होता है।

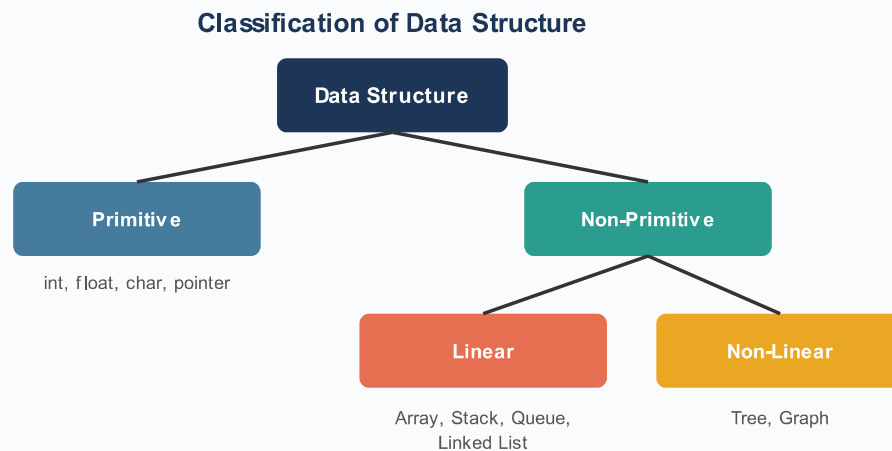
Is Unit ki Structure:

- 1) Classification of Data Structure — Linear, Non-Linear, Primitive, Non-Primitive
- 2) Pointers — Introduction, Declaration, Pointer Arithmetic, Array of Pointers, 2-D Array Implementation
- 3) Structure — Definition, Declaration, Initializing
- 4) Operations on Data Structures — Traversing, Searching, Insertion, Deletion, Sorting, Merging, Updating

1. Classification of Data Structure

Data Structures को उनके properties aur nature के आधार पर मुख्य रूप से दो ways में classify किया जाता है — पहला **Primitive vs Non-Primitive** के आधार पर, aur दूसरा **Linear vs Non-Linear** के आधार पर।

- Classification को समझने से यह decide करना आसान हो जाता है कि किसी specific problem के लिए कौन सी data structure सबसे उपयुक्त रहेगी।
- हर type की data structure का अपना एक अलग storage तरीका aur अलग operations होते हैं।



चित्र 1.1 — Data Structure पहले Primitive aur Non-Primitive में divide होता है, फिर Non-Primitive को आगे Linear aur Non-Linear में divide किया जाता है।

1.1 Primitive Data Structure

Primitive Data Structures वे basic data types होते हैं जो directly programming language द्वारा support किए जाते हैं aur इन्हें आगे और तोड़ा (break) नहीं जा सकता। यह data structures का सबसे बुनियादी (fundamental) level है।

- यह single value store करते हैं, जैसे एक integer number या एक character।
- Examples: **int, float, char, double, pointer**।
- यह directly CPU द्वारा operate किए जा सकते हैं, इसलिए इनकी processing सबसे fast होती है।

1.2 Non-Primitive Data Structure

Non-Primitive Data Structures वे data structures होते हैं जो primitive data types को combine करके बनाए जाते हैं, ताकि complex aur बड़ी मात्रा में data को organize किया जा सके।

- यह एक से ज़्यादा values को एक साथ, एक meaningful तरीके से store कर सकते हैं।
- Examples: Array, Linked List, Stack, Queue, Tree, Graph।
- इन्हें आगे **Linear** aur **Non-Linear** में divide किया जाता है।

1.3 Linear Data Structure

Linear Data Structure वह होती है जिसमें सभी elements एक **sequential (क्रमबद्ध) order** में arranged होते हैं, यानी हर element का सिर्फ एक ही previous aur एक ही next element होता है।

- Elements को linearly traverse (एक के बाद एक) किया जा सकता है, single level में।
- Memory में यह एक sequence की तरह store होते हैं (चाहे contiguous हों जैसे array, या non-contiguous जैसे linked list)।

- Examples: **Array, Stack, Queue, Linked List**।

1.4 Non-Linear Data Structure

Non-Linear Data Structure वह होती है जिसमें elements एक sequential order में नहीं, बल्कि **hierarchical (पदानुक्रमित)** या **network जैसी** structure में arranged होते हैं।

- एक element के एक से ज़्यादा previous या next elements हो सकते हैं (जैसे tree में parent-children relationship)।
- Elements को traverse करने के multiple रास्ते हो सकते हैं।
- Examples: **Tree, Graph**।

Basis	Linear Data Structure	Non-Linear Data Structure
Arrangement	Sequential (एक के बाद एक)	Hierarchical या interconnected
Traversal	Single pass में possible	Multiple levels/paths में करना पड़ता है
Memory Utilization	अक्सर कम efficient (जैसे array में)	ज़्यादा efficient हो सकता है
Examples	Array, Stack, Queue, Linked List	Tree, Graph
Complexity	Implement करना अपेक्षाकृत आसान	Implement करना ज़्यादा जटिल

1.5 Features

- हर classification की अपनी specific storage aur access pattern होती है, जो अलग-अलग problems के लिए उपयुक्त होती है।
- Primitive data structures memory में directly store होती हैं, जबकि non-primitive उन्हें combine करके बनाई जाती हैं।
- Linear structures समझना aur implement करना आसान होते हैं, इसलिए beginners पहले इन्हें सीखते हैं।
- Non-linear structures real-world की complex relationships (जैसे family tree, social network) को represent करने में बेहतर होते हैं।

Advantages of Proper Classification

- Problem के according सही data structure चुनना आसान हो जाता है।
- Memory उपयोग aur processing speed को optimize करने में मदद मिलती है।
- Code को organize aur maintain करना आसान हो जाता है।

Challenges

- गलत data structure चुनने से program की performance खराब हो सकती है।
- Non-linear structures को समझना aur implement करना beginners के लिए मुश्किल हो सकता है।

Example: किसी queue में लगे लोगों को represent करने के लिए Linear structure (Queue) उपयुक्त है, जबकि किसी company के organizational hierarchy (CEO → Managers → Employees) को represent करने के लिए Non-Linear structure (Tree) उपयुक्त है।

2. Pointers

2.1 Introduction to Pointers

Pointer एक ऐसा special variable होता है जो किसी दूसरे variable की **memory address** को store करता है, ना कि उस variable की actual value को। आसान भाषा में कहें तो, pointer एक "address book entry" की तरह है, जो बताता है कि actual data memory में कहाँ रखा हुआ है।

- हर variable को computer memory में एक unique address मिलता है, aur pointer उसी address को अपने अंदर store करता है।
- Pointers की मदद से memory को directly access aur manipulate किया जा सकता है, जो programs को ज़्यादा efficient बनाता है।
- Pointers का उपयोग dynamic memory allocation, arrays, functions, aur data structures (जैसे linked list) बनाने में बहुत होता है।

Pointer — Basic Concept



चित्र 2.1 — Pointer variable अपने अंदर value नहीं, बल्कि किसी दूसरे variable का memory address store करता है।

2.2 Declaring and Initializing Pointers

Pointer को declare करते समय उसके data type के आगे **asterisk (*)** symbol लगाया जाता है, जो बताता है कि यह variable एक pointer है।

```
int num = 10;          // normal integer variable
int *ptr;              // pointer declaration
ptr = #                // pointer initialization (& = address-of
operator)

printf("%d", *ptr);    // *ptr = value at that address = 10
```

- **Declaration** — किसी variable को pointer के रूप में define करना (जैसे `int *ptr;`)।
- **Initialization** — pointer को किसी variable के address से assign करना (जैसे `ptr = #`)।
- **& (Address-of Operator)** — किसी variable का memory address निकालने के लिए इस्तेमाल होता है।
- *** (Dereference Operator)** — pointer के through उस address पर मौजूद actual value को access करने के लिए इस्तेमाल होता है।

2.3 Accessing Variables using Pointers

Pointer के through किसी variable की value को access करना **Dereferencing** कहलाता है — यानी pointer में store address पर जाकर वहाँ मौजूद actual data को पढ़ना या बदलना।

```
int num = 25;
int *ptr = #

printf("Value: %d\n", *ptr); // Output: 25 (dereferencing)
*ptr = 50; // pointer के through value
change की
printf("New Value: %d\n", num); // Output: 50 (num खुद भी बदल
गया)
```

- Pointer के through value बदलने से actual original variable की value भी बदल जाती है, क्योंकि दोनों एक ही memory location को refer करते हैं।
- यह concept function calls में **Call by Reference** के लिए बहुत महत्वपूर्ण है।

2.4 Pointer Arithmetic

Pointer Arithmetic का मतलब है pointer variables पर mathematical operations (जैसे addition, subtraction) perform करना, ताकि memory में एक location से दूसरी location पर move किया जा सके।

Operation	Description
Increment (ptr++)	Pointer को अगली memory location (data type के size जितना आगे) पर move करता है
Decrement (ptr--)	Pointer को पिछली memory location पर move करता है

Addition (ptr + n)	Pointer को n locations आगे move करता है
Subtraction (ptr - n)	Pointer को n locations पीछे move करता है
Pointer Difference (ptr2 - ptr1)	दो pointers के बीच elements की संख्या बताता है

Yaad rakhein: जब कोई integer pointer (`int *ptr`) increment किया जाता है, तो वह actual memory address में 1 नहीं, बल्कि `int` के size (आमतौर पर 4 bytes) जितना आगे बढ़ता है — यह data type के according automatically adjust होता है।

2.5 Array of Pointers

Array of Pointers एक ऐसा array होता है जिसका हर element खुद एक pointer होता है, यानी हर element किसी दूसरे variable या data का address store करता है।

```
int a = 10, b = 20, c = 30;
int *arr[3];      // array of 3 integer pointers

arr[0] = &a;
arr[1] = &b;
arr[2] = &c;

printf("%d", *arr[1]); // Output: 20
```

- यह concept खासतौर पर strings के arrays बनाने में बहुत उपयोगी है (जैसे character pointers का array)।
- Array of pointers की मदद से अलग-अलग size या location की memory को एक साथ manage करना आसान हो जाता है।

2.6 Row-Major aur Column-Major Implementation of 2-D Array

2-D array को computer की memory में actually एक **linear (1-D) sequence** की तरह ही store किया जाता है, क्योंकि memory खुद linear होती है। इसके लिए दो तरीके इस्तेमाल होते हैं — **Row-Major Order** aur **Column-Major Order**।

Row-Major vs Column-Major Storage

2-D Array (2x3)

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]

Same Array

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]

Row-Major (पहले पूरी row, फिर अगली row)

A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] A[1][2]

Column-Major (पहले पूरा column, फिर अगला column)

A[0][0] A[1][0] A[0][1] A[1][1] A[0][2] A[1][2]

दोनों तरीकों में memory linear (1-D) ही होती है, बस elements को store करने का order अलग है

चित्र 2.2 — Row-Major order में एक row के सभी elements एक साथ store होते हैं, जबकि Column-Major order में एक column के सभी elements एक साथ store होते हैं।

Basis

Row-Major Order

Column-Major Order

Storage Order	पहले पहली row के सभी elements, फिर दूसरी row	पहले पहले column के सभी elements, फिर दूसरा column
Used in Languages	C, C++, Java जैसी languages में	Fortran, MATLAB जैसी languages में
Address Formula	$\text{Address} = \text{Base} + (i \times \text{NumCols} + j) \times \text{size}$	$\text{Address} = \text{Base} + (j \times \text{NumRows} + i) \times \text{size}$

2.7 Features of Pointers

- Pointers memory को direct access करने की सुविधा देते हैं, जिससे programs ज़्यादा efficient बनते हैं।
- Dynamic memory allocation (जैसे runtime में memory लेना) सिर्फ pointers की मदद से ही possible होता है।
- Functions में large data structures (जैसे arrays) को बिना copy किए pass करने के लिए pointers इस्तेमाल होते हैं।
- Linked List, Tree, Graph जैसी data structures बनाने के लिए pointers अनिवार्य (mandatory) होते हैं।

Advantages

- Memory का efficient उपयोग करना possible बनाते हैं।
- Functions में data को Call by Reference से pass करना आसान बनाते हैं।
- Dynamic data structures (linked list, tree) बनाने के लिए ज़रूरी हैं।
- Arrays aur strings को efficiently manipulate करने में मदद करते हैं।

Disadvantages

- गलत इस्तेमाल से program crash हो सकता है (जैसे invalid memory access)।
- Pointers को समझना aur debug करना beginners के लिए मुश्किल हो सकता है।
- Memory leaks aur dangling pointers जैसी समस्याएँ आ सकती हैं अगर सही से manage न किया जाए।
- Code को पढ़ना aur समझना pointers की वजह से कभी-कभी जटिल हो जाता है।

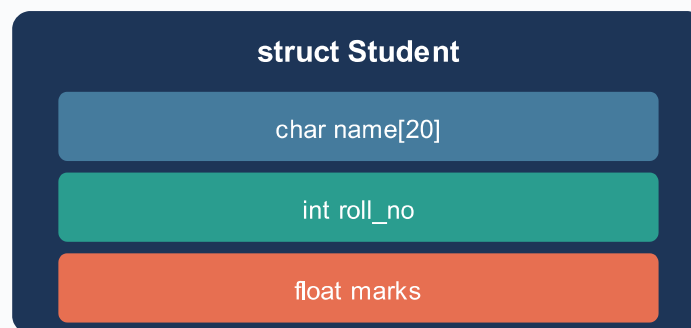
Example: जब कोई function किसी बड़े array को receive करता है, तो पूरा array copy करने की बजाय उसका सिर्फ pointer (address) pass किया जाता है, जिससे memory aur processing time दोनों बचते हैं।

3. Structure

Structure C programming का एक user-defined data type है, जो अलग-अलग data types (जैसे int, float, char) के variables को एक साथ, एक ही नाम के under group करने की सुविधा देता है। आसान भाषा में कहें तो, अगर किसी real-world entity (जैसे student) की कई properties (नाम, roll number, marks) को एक साथ store करना हो, तो structure इस्तेमाल किया जाता है।

- Structure के अंदर मौजूद हर variable को **member** कहा जाता है, aur यह अलग-अलग data types के हो सकते हैं।
- Array सिर्फ same data type के elements store कर सकता है, लेकिन structure अलग-अलग data types को एक साथ group कर सकता है — यही इसकी सबसे बड़ी खासियत है।

Structure — Memory Layout



चित्र 3.1 — एक Structure के अंदर अलग-अलग data types (char, int, float) के members एक साथ group होकर एक ही unit की तरह store होते हैं।

3.1 Definition

```
struct Student {  
    char name[20];  
    int roll_no;  
    float marks;  
};
```

यहाँ `struct Student` एक नया data type define करता है, जिसमें तीन members हैं — `name` , `roll_no` , aur `marks` ।

3.2 Declaration

```
struct Student s1;    // structure variable declare करना  
struct Student s2, s3; // एक साथ multiple variables declare करना
```

Declaration करने के बाद ही उस structure type की actual memory allocate होती है, जिसे variables के through access किया जा सकता है।

3.3 Initializing Structure

```
struct Student s1 = {"Aman", 101, 89.5};  
  
// या member के through individually assign करना:  
strcpy(s1.name, "Aman");  
s1.roll_no = 101;  
s1.marks = 89.5;  
  
// members को access करने के लिए dot (.) operator इस्तेमाल होता है  
printf("%s", s1.name);
```

- Structure के members को access करने के लिए **dot (.) operator** इस्तेमाल होता है।

- अगर structure का pointer हो, तो members को access करने के लिए **arrow (-> operator)** इस्तेमाल होता है।

3.4 Features

- अलग-अलग data types के variables को एक single unit में group करता है।
- Real-world entities (जैसे student, employee, book) को logically represent करना आसान बनाता है।
- Structures के array भी बनाए जा सकते हैं, जिससे multiple records को manage करना आसान हो जाता है।
- Nested structures possible हैं, यानी एक structure के अंदर दूसरी structure भी हो सकती है।

Advantages

- Related data को logically एक साथ group करके code को organize रखता है।
- Complex real-world data को represent करना आसान बनाता है।
- Functions में पूरे structure को एक साथ pass करना possible बनाता है।
- Code की readability aur maintainability बढ़ जाती है।

Disadvantages

- अगर structure बड़ा है, तो उसे pass/copy करने में ज़्यादा memory aur time लग सकता है।
- Structures में built-in methods/functions नहीं होते (जैसे classes में होते हैं)।
- Memory alignment की वजह से structures कभी-कभी ज़रूरत से ज़्यादा memory ले सकते हैं (padding)।

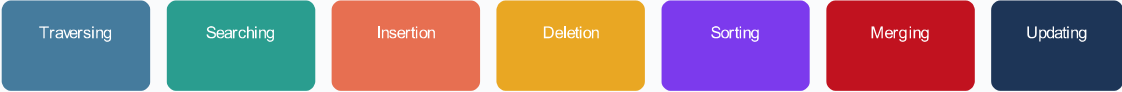
Example: किसी college के database program में हर student की जानकारी (name, roll number, marks) को store करने के लिए एक Student structure

बनाया जाता है, जिससे सारी related information एक साथ manage हो सके।

4. Operations on Data Structures

Data Structures पर कई common operations perform की जाती हैं, ताकि data को effectively manage किया जा सके — जैसे data को ढूँढना, नया data जोड़ना, पुराना data हटाना, आदि।

Common Operations on Data Structures



चित्र 4.1 — Data Structures पर perform की जाने वाली 7 मुख्य operations।

Operation	Description
Traversing	Data structure के हर element को एक-एक करके visit करना, ताकि उन्हें process (जैसे print) किया जा सके
Searching	Data structure में किसी specific element को ढूँढना (जैसे Linear Search, Binary Search)

Insertion	Data structure में एक नया element जोड़ना, चाहे शुरुआत में, बीच में, या अंत में
Deletion	Data structure से किसी मौजूदा element को हटाना
Sorting	Elements को किसी particular order (ascending/descending) में arrange करना
Merging	दो अलग-अलग data structures (जैसे दो sorted arrays) को combine करके एक नया बनाना
Updating	Data structure में किसी मौजूदा element की value को बदलना

4.1 Traversing

Traversing का मतलब है data structure के हर element को systematically, एक-एक करके visit करना, ताकि हर element पर कोई operation (जैसे display करना) perform किया जा सके।

```
for(i = 0; i < n; i++) {  
    printf("%d ", arr[i]);    // हर element को traverse करके  
    print करना  
}
```

4.2 Searching

Searching का मतलब है data structure में किसी specific element को ढूँढना, यह check करते हुए कि वह मौजूद है या नहीं, aur अगर है तो उसकी location क्या है।

- **Linear Search** — हर element को एक-एक करके check करना, जब तक desired element न मिल जाए।
- **Binary Search** — sorted data structure में, हर बार आधे data को ignore करते हुए, जल्दी search करना।

4.3 Insertion

Insertion का मतलब है data structure में कोई नया element add करना — यह शुरुआत में, बीच में, या अंत में किया जा सकता है, data structure के type पर निर्भर करता है।

4.4 Deletion

Deletion का मतलब है data structure से किसी मौजूदा element को permanently हटाना, aur बाकी elements को (ज़रूरत पड़ने पर) सही position में re-arrange करना।

4.5 Sorting

Sorting का मतलब है elements को किसी specific क्रम (order) में arrange करना, आमतौर पर **ascending** (छोटे से बड़े) या **descending** (बड़े से छोटे) order में। Bubble Sort, Selection Sort, Insertion Sort इसके common examples हैं।

4.6 Merging

Merging का मतलब है दो अलग-अलग data structures (आमतौर पर दो sorted lists/arrays) को combine करके एक नई, बड़ी data structure बनाना, जिसमें दोनों के elements शामिल हों (अक्सर sorted order बनाए रखते हुए)।

4.7 Updating

Updating का मतलब है data structure में पहले से मौजूद किसी element की value को नई value से बदलना, बिना उसकी position या बाकी elements को प्रभावित किए।

4.8 Features of these Operations

- हर operation की अपनी अलग time complexity होती है, जो data structure के type पर निर्भर करती है।
- Sorted data structures में searching (जैसे Binary Search) unsorted की तुलना में काफी तेज़ होती है।
- Insertion aur Deletion की efficiency data structure के type के according अलग-अलग होती है (जैसे Linked List में यह Array से आसान होती है)।
- यह सभी operations मिलकर किसी भी program में data को effectively manage करना possible बनाती हैं।

Advantages

- Data को efficiently manage aur manipulate करना possible बनाती हैं।
- अलग-अलग operations के through data की accuracy aur relevance बनाए रखी जा सकती है।
- Searching aur sorting operations, data को जल्दी access करने में मदद करते हैं।

Disadvantages

- कुछ operations (जैसे Array में insertion/deletion) time-consuming हो सकते हैं, क्योंकि elements को shift करना पड़ता है।
- गलत data structure चुनने पर operations की performance काफी खराब हो सकती है।
- Sorting aur merging जैसी operations बड़े data पर ज़्यादा memory aur processing time ले सकती हैं।

Example: किसी phone की contact list में नया contact जोड़ना (Insertion), किसी नाम को ढूँढना (Searching), पुराने contact को हटाना (Deletion), aur सभी contacts को alphabetically arrange करना (Sorting) — यह सब data structure operations के real-life examples हैं।

अगर आप पूरा **PDF** खरीदना चाहते हैं
जिसकी कीमत केवल ~~120~~ **79** रू है
तो नीचे दिये गये WhatsApp नंबर पर संपर्क करें -
9009300541, 9144978989

Notesinhindi.com

NotesInHindi