**What is Data Structure 🡪**

**A Data Structure is a way of organizing and storing data so that it can be accessed and modified efficiently.**

**For example 🡪 storing medicine in medical store in a organized form, books in libraries**

**Think of it as a container or format that helps manage large amounts of data effectively for operations like:**

* **Searching**
* **Insertion**
* **Deletion**
* **Sorting**
* **Traversal**

**Why are Data Structures Important🡪**

Efficient **storage** and **retrieval** of data.

Helps in **optimizing time** and **space** complexity.

Forms the **foundation of algorithms** and system design.

Crucial for solving **real-world problems** like social media feeds, navigation systems, etc.

# Type of Data Structure 🡪

1. Primitive Data type 🡪

* **Primitive Data Types** are the **basic building blocks** of any programming language. from which you build larger structures (like data structures or programs).
* They are **predefined** by the language and are used to **store simple values** like numbers, characters, or true/false.
* are used to hold **one value at a time**, like:

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| int | Stores whole numbers | int age = 20; |
| float | Stores decimal numbers | float pi = 3.14; |
| char | Stores a single character | char grade = 'A'; |
| bool / \_Bool | Stores true/false values (C: \_Bool) | bool isOn = true; |
| double | Stores large decimal numbers | double pi = 3.141592; |
| void | Means "no value"/empty (used in functions) | void display(); |

| **Data Type** | **Size (Bytes)** |
| --- | --- |
| char | 1 byte |
| int | 4 bytes |
| float | 4 bytes |
| double | 8 bytes |

Example 🡪

int age = 21;

char grade = 'A';

**Non-Primitive Data Structures 🡪**

* These are derived from primitive types and can store multiple values.

These are two types Linear and Non Linear

a)Linear 🡪

Elements are arranged in a sequential manner.

| Type | Description | Example |
| --- | --- | --- |
| Array | Fixed-size, same data type | int arr[5] = {1,2,3,4,5} |
| Linked List | Nodes linked with pointers | struct Node { int data; struct Node \*next; } |
| Stack | LIFO (Last In First Out) | Undo operation in editors |
| Queue | FIFO (First In First Out) | Print queue in printer |
| **B. Non-Linear Data Structures**  Elements are **not arranged sequentially**.   | **Type** | **Description** | **Example** | | --- | --- | --- | | Tree | Hierarchical, root-node based | Binary Tree, BST, Heap | | Graph | Nodes (vertices) connected by edges | Social networks, maps | |  |  |

Applications of Data Structures

| Area | Use Case |
| --- | --- |
| OS | Process scheduling using queues |
| DBMS | Indexing using B-Trees |
| Web Browsers | Back/Forward history using stack |
| AI & Machine Learning | Graphs, trees, heaps |
| Social Media | Graphs to represent user connections |

**Installation 🡪 start with c programming**

* <https://installc.org/> 🡪 installing c
* Install c/c++ extension
* <https://www.msys2.org/> 🡪 download mingW
* After install Open **MSYS2 terminal** and run the following commands:
* pacman -Syu # update system (close and reopen terminal after)
* pacman -S mingw-w64-x86\_64-gcc # install GCC

**Alternative 3: Use TDM-GCC (Simple Installer)**

1. Go to: https://jmeubank.github.io/tdm-gcc/
2. Download the installer.
3. Install it — it will set up GCC automatically.

Then, use a simple editor like **Notepad++** or **VS Cod**

or try online compiler of C using

<https://www.programiz.com/c-programming/online-compiler>

<https://www.onlinegdb.com>

<https://onecompiler.com/c>

# First Code 🡪

#include <stdio.h>        // 1. Include the standard input-output library

int main() {              // 2. Starting point of your program

    printf("Hello, World!");  // 3. Print message to the screen

    return 0;             // 4. End the program successfully

}

#include<stdio.h>

int main() {

int age = 20;

float pi = 3.14;

char grade = 'A';

char name[] = "Shubham";

printf("Age: %d\n", age); //for printing itegers

printf("Pi: %f\n", pi); //for printing float

printf("Grade: %c\n", grade); // for printing single character

printf("Name: %s\n", name); //for printing string

return 0;

}

| Line | Explanation |
| --- | --- |
| #include <stdio.h> | Tells the compiler to include the Standard Input Output library, which allows you to use printf() and scanf() |
| int main() | This is the main function, where program execution starts |
| { ... } | Curly braces enclose the body of the main function |
| printf("Hello, World!"); | Prints the message to the console |
| return 0; | Tells the system that the program finished successfully |
|  |  |

Homogeneous Data🡪

Data items that are of the same type are called homogeneous data.

int numbers[] = {10, 20, 30, 40};

char \*arr[] = {"10", "Hello", "20", "World"};

Heterogeneous Data🡪

Data items that are of different types are called heterogeneous data.

struct Student {

int rollNo;

char name[20];

float marks;

};

Now start with first Data Structure 🡪

1)Array 🡪 An Array is a data structure that stores collection of elements of same type stored at contiguous memory locations and can be accessed using an index

Datatype arrayName [array Size] 🡪

Int num[5] = {2,8,7,6,0} or

Int num[] = {2,8,7,6,0}

a)Change the value of array elements 🡪

num[2] = 31 it will directly change the value of 2nd index

Advantages or Array 🡪

* Efficient storage and retrieval 🡪 array store elements in contiguous memory location, which makes it easy to retrieve elemnts using their index. So very efficient with large amount of data
* Random access(fast access): array allow access to individual elements using their index , which means that accessing any elemnt of array takes same amount of time
* Easy to sort and research: Arrays can be easily sorted and searched using algorithms like binary serach , which can be more efficient than searching through unsorted data
* Flexibility: array can be used to represent a wide variety of data structure including stacks , queues etc

Disadvantage 🡪

1. FixedSize:In most programming languages array have a fixed size that can not be changed once they are created . this can make it difficult to work with data structure that need to grow or shrink dynamically

b)No built in support for insertion or deletion : inserting or deleting an element in an array can be time consuming and require shifting all the elements after the insertion or deletion point

c)Homogeneous elemnts: Array can only store elemnts od the same type which can be limiting for many requirements.

d) poor performance for some operations: some operations , such as searching or inserting elements in a sorted array, can have poor performance compared to other structures like hash tables or binary search trees

lowerBound 🡪 starting index

upperBound 🡪 ending index

get number of elemnts = (upperBound – lowerBound) +1

each index consumes 4 bytes

Size of array = numberOf Elemnts \* size of each elemnts in byts

= 10 \* 4 = 40B

Find Address of the element at kth index 🡪

A[k] = B +W\*(k -lowerBound)

b🡪 is the base address suppose array address start with 1200 then 1204, 1208..

W🡪 is the size of each elemnt 🡪 3B or 4B

K 🡪 is the index of elemnt

lowerBound 🡪 first Index of element

upper Bound 🡪 last Index of elemnt

int arr[] = {2,4,7,10,8,1} I want to find the address of 10

a[10] = 1200 + 4\*(3 - 0) // 1212

traverse,Insertion, deletion, searching, sorting 🡪

1)Traverse 🡪

#include<stdio.h>

// traverse example

int main(){

int size = 5;

int arr[5] = {3,2,5,0,2};

for(int i = 0; i < size; i++) {

if (i == size - 1)

printf("%d", arr[i]); // no space after last element

// printf("%d\n", arr[i]); // no space after last element

else

printf("%d ", arr[i]);

}

return 0;

}

2) Inserstion

#include <stdio.h>

int main() {

int arr[5] = {10, 20, 30, 40}; // only 4 elements initialized

int pos = 2, val = 12;

// Shift elements right to make space

for (int i = 4; i > pos; i--) {

arr[i] = arr[i-1];

}

// Insert the value

arr[pos] = val;

// Print the array

for (int i = 0; i < 5; i++) {

printf("%d ", arr[i]);

}

return 0;

}

1. Deletion of value 🡪

#include<stdio.h>

int main(){

int arr[5] = {2,4,9,8,10};

// int arr[5] = {2,4,9,8,0};

// int arr[5] = {2,4,0,9,8};

int pos = 2 , n=5;

for(int i=pos; i<n; i++){

arr[i] = arr[i+1];

}

n--;

for(int i=0; i<n; i++){

printf("%d ", arr[i]);

}

}

3)search (Linear search)🡪

int key = 30, found = 0;

for(int i = 0; i < n; i++) {

if(arr[i] == key) {

printf("Found at index %d", i);

found = 1;

break;

}

}

if(!found) printf("Not found");