Data structures in c

What is a data structure

Classification of data structures

Popular types of data structures

What is data structure?

A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

A data structure is not only used for organizing the data. It is also used for processing, retrieving, and storing data. There are different basic and advanced types of data structures that are used in almost every program or software system that has been developed. So we must have good knowledge about data structures.

Classification of data structures



Linear data structure: Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.   
Examples of linear data structures are array, stack, queue, linked list, etc.

Static data structure: Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.   
An example of this data structure is an array.

Dynamic data structure: In dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.   
Examples of this data structure are queue, stack, etc.

Non-linear data structure: Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can’t traverse all the elements in a single run only.   
Examples of non-linear data structures are trees and graphs.

For example, we can store a list of items having the same data-type using the array data structure.



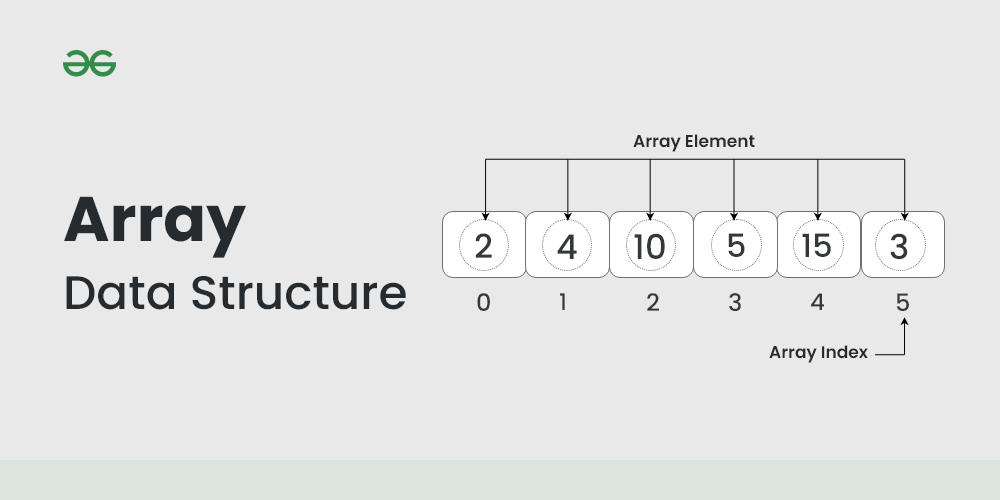
Popular types of data structures

1. [Array](https://www.geeksforgeeks.org/array/)
2. [Linked List](https://www.geeksforgeeks.org/data-structures/linked-list/)
3. [Stack](https://www.geeksforgeeks.org/stack/)
4. [Queue](https://www.geeksforgeeks.org/queue/)
5. [Binary Tree](https://www.geeksforgeeks.org/binary-tree-2/)
6. [Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree/)
7. [Heap](https://www.geeksforgeeks.org/heap/)
8. [Hashing](https://www.geeksforgeeks.org/hashing/)
9. [Graph](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/)
10. [Matrix](https://www.geeksforgeeks.org/matrix/)
11. [Misc](https://www.geeksforgeeks.org/data-structures/" \l "Misc)
12. [Advanced Data Structure](https://www.geeksforgeeks.org/advanced-data-structures/)

Array data structure

What is an array?

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).



The above image can be looked as a top-level view of a staircase where you are at the base of the staircase. Each element can be uniquely identified by their index in the array (in a similar way as you could identify your friends by the step on which they were on in the above example).

Operations

1. Declaration:

To declare an array in c, you specify the data type of its elements and the array’s name, followed by its size in square brackets



1. Initialization:

You can initialize an array at the time of declaration



If the size is omitted, the compiler will infer it from the number of elements in the initialization list



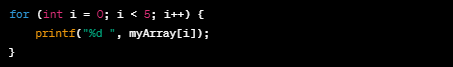
1. Accessing elements:

Elements in an array can be accessed using their index. Array indices start from 0



Array operations:

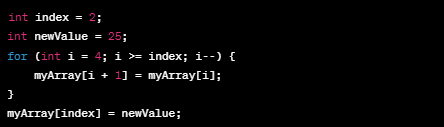
1. Traversal: use loops for traversing the entire array



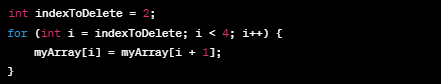
1. Insertion and deletion:

C arrays have a fixed size, so insertion and deletion require manual shifting. The following are examples of insertion and deletion respectively

insertion



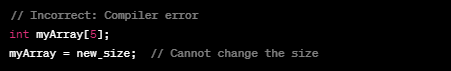
Deletion



Important considerations

1. Array size:

Once declared, the size of an array in c cannot be changed



1. Bound checking:

C does not perform array bounds checking by default. It’s the programmers responsibility to ensure that array indices stay within bounds



If forinstance the myArray array has an upper bound of less than 10

1. Multidimensional arrays:

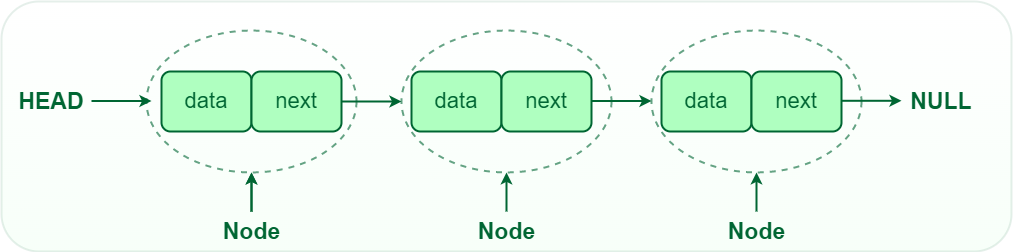
C supports multidimensional arrays



Arrays in c are powerful but require careful management to avoid common pitfalls like buffer overflows and index out-of-bound errors. A buffer overflow is a type of software vulnerability that occurs when a program writes more data to a block of memory, or buffer than it was allocated for. This excess data can overflow into adjacent memory, potentially overwritting other data or code and leading to unpredictable behavior, crashes or security vulnerabilities.

Linked list data structure

Linked List is a linear data structure, in which elements are not stored at a contiguous location, rather they are linked using pointers. Linked List forms a series of connected nodes, where each node stores the data and the address of the next node.



Node Structure: A node in a linked list typically consists of two components:  
Data: It holds the actual value or data associated with the node.  
Next Pointer: It stores the memory address (reference) of the next node in the sequence.  
Head and Tail: The linked list is accessed through the head node, which points to the first node in the list. The last node in the list points to NULL or nullptr, indicating the end of the list. This node is known as the tail node.

Why linked list data structure needed?

Here are a few advantages of a linked list that is listed below, it will help you understand why it is necessary to know.

Dynamic Data structure: The size of memory can be allocated or de-allocated at run time based on the operation insertion or deletion.

Ease of Insertion/Deletion: The insertion and deletion of elements are simpler than arrays since no elements need to be shifted after insertion and deletion, Just the address needed to be updated.

Efficient Memory Utilization: As we know Linked List is a dynamic data structure the size increases or decreases as per the requirement so this avoids the wastage of memory.

Implementation: Various advanced data structures can be implemented using a linked list like a stack, queue, graph, hash maps, etc.

Example:

In a system, if we maintain a sorted list of IDs in an array id[] = [1000, 1010, 1050, 2000, 2040].

If we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved due to this so much work is being done which affects the efficiency of the code.

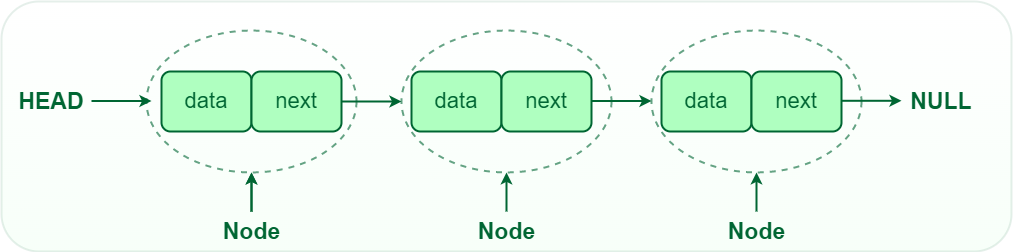
[Types of linked lists](https://www.geeksforgeeks.org/types-of-linked-list/):

There are mainly three types of linked lists:

* Single-linked list
* Double linked list
* Circular linked list

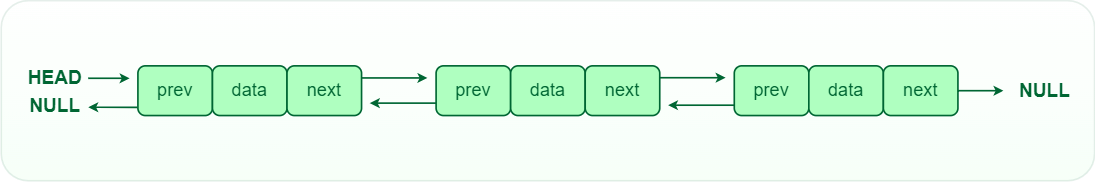
1. [Single-linked list](https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/):

In a singly linked list, each node contains a reference to the next node in the sequence. Traversing a singly linked list is done in a forward direction.



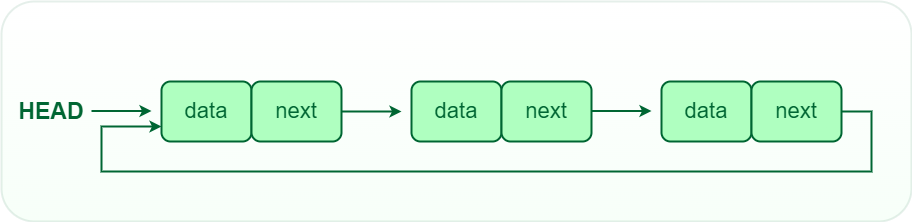
2. [Double-linked list](https://www.geeksforgeeks.org/introduction-and-insertion-in-a-doubly-linked-list/):

In a doubly linked list, each node contains references to both the next and previous nodes. This allows for traversal in both forward and backward directions, but it requires additional memory for the backward reference.



3. [Circular linked list](https://www.geeksforgeeks.org/circular-linked-list/):

 In a circular linked list, the last node points back to the head node, creating a circular structure. It can be either singly or doubly linked.



Operations on Linked Lists

[Insertion](https://www.geeksforgeeks.org/insertion-in-linked-list/): Adding a new node to a linked list involves adjusting the pointers of the existing nodes to maintain the proper sequence. Insertion can be performed at the beginning, end, or any position within the list

[Deletion](https://www.geeksforgeeks.org/deletion-in-linked-list/): Removing a node from a linked list requires adjusting the pointers of the neighboring nodes to bridge the gap left by the deleted node. Deletion can be performed at the beginning, end, or any position within the list.

[Searching](https://www.geeksforgeeks.org/search-an-element-in-a-linked-list-iterative-and-recursive/): Searching for a specific value in a linked list involves traversing the list from the head node until the value is found or the end of the list is reached.

Advantages of Linked Lists

Dynamic Size: Linked lists can grow or shrink dynamically, as memory allocation is done at runtime.

Insertion and Deletion: Adding or removing elements from a linked list is efficient, especially for large lists.

Flexibility: Linked lists can be easily reorganized and modified without requiring a contiguous block of memory.

Disadvantages of Linked Lists

Random Access: Unlike arrays, linked lists do not allow direct access to elements by index. Traversal is required to reach a specific node.

Extra Memory: Linked lists require additional memory for storing the pointers, compared to arrays.