**Stack** is a linear data structure that follows a particular order in which the operations are performed. The order may be **LIFO(Last In First Out)** or **FILO(First In Last Out)**.

**Basic Operations on Stack**

In order to make manipulations in a stack, there are certain operations provided to us.

* **push()** to insert an element into the stack
* **pop()**to remove an element from the stack
* **top()** Returns the top element of the stack.
* **isEmpty()**returns true if stack is empty else false.
* **size()** returns the size of stack.

## ****Push:****

Adds an item to the stack. If the stack is full, then it is said to be an**Overflow condition.**

## ****Pop:****

Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an **Underflow** **condition.**

## ****Top:****

Returns the top element of the stack.

## ****isEmpty:****

Returns true if the stack is empty, else false.

**Complexity Analysis:**

* **Time Complexity**

| **Operations** | **Complexity** |
| --- | --- |
| push() | O(1) |
| pop() | O(1) |
| isEmpty() | O(1) |
| size() | O(1) |

In addition to these two main types, there are several other variations of Stacks, including:

1. **Infix to Postfix Stack**: This type of stack is used to convert infix expressions to postfix expressions.
2. **Expression Evaluation Stack**: This type of stack is used to evaluate postfix expressions.
3. **Recursion Stack**: This type of stack is used to keep track of function calls in a computer program and to return control to the correct function when a function returns.
4. **Memory Management Stack**: This type of stack is used to store the values of the program counter and the values of the registers in a computer program, allowing the program to return to the previous state when a function returns.
5. **Balanced Parenthesis Stack**: This type of stack is used to check the balance of parentheses in an expression.
6. **Undo-Redo Stack**: This type of stack is used in computer programs to allow users to undo and redo actions.

## ****Applications of the stack:****

* [Infix to Postfix](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/) /Prefix conversion
* Redo-undo features at many places like editors, photoshop.
* Forward and backward features in web browsers
* Used in many algorithms like [Tower of Hanoi,](https://www.geeksforgeeks.org/recursive-functions/)[tree traversals](https://www.geeksforgeeks.org/618/), [stock span problems](https://www.geeksforgeeks.org/the-stock-span-problem/), and [histogram problems](https://www.geeksforgeeks.org/largest-rectangular-area-in-a-histogram-set-1/).
* Backtracking is one of the algorithm designing techniques. Some examples of backtracking are the Knight-Tour problem, N-Queen problem, find your way through a maze, and game-like chess or checkers in all these problems we dive into someway if that way is not efficient we come back to the previous state and go into some another path. To get back from a current state we need to store the previous state for that purpose we need a stack.
* In Graph Algorithms like [Topological Sorting](https://www.geeksforgeeks.org/topological-sorting/) and [Strongly Connected Components](https://www.geeksforgeeks.org/strongly-connected-components/)
* In Memory management, any modern computer uses a stack as the primary management for a running purpose. Each program that is running in a computer system has its own memory allocations
* String reversal is also another application of stack. Here one by one each character gets inserted into the stack. So the first character of the string is on the bottom of the stack and the last element of a string is on the top of the stack. After Performing the pop operations on the stack we get a string in reverse order.
* Stack also helps in implementing function call in computers. The last called function is always completed first.
* Stacks are also used to implement the undo/redo operation in text editor.

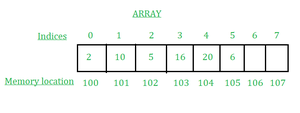
## [****What is Array?****](https://www.geeksforgeeks.org/introduction-to-arrays/)

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).

### **Time complexity:**

|  |  |  |
| --- | --- | --- |
| **Algo.** | **Avg case** | **worst case** |
| accessing | O(1) | O(1) |
| searching | O(n) | O(n) |
| insertion | O(n) | O(n) |
| deletion | O(n) | O(n) |

[**Array**](https://www.geeksforgeeks.org/array-data-structure/) is a linear data structure that is a collection of similar data types. Arrays are stored in contiguous memory locations. It is a static data structure with a fixed size. It combines data of similar types.



***ARRAY***

**Applications of Array Data Structure:**

Below are some applications of arrays.

* **Storing and accessing data**: Arrays are used to store and retrieve data in a specific order. For example, an array can be used to store the scores of a group of students, or the temperatures recorded by a weather station.
* **Sorting:** Arrays can be used to sort data in ascending or descending order. Sorting algorithms such as bubble sort, merge sort, and quicksort rely heavily on arrays.
* **Searching**: Arrays can be searched for specific elements using algorithms such as linear search and binary search.
* **Matrices**: Arrays are used to represent matrices in mathematical computations such as matrix multiplication, linear algebra, and image processing.
* **Stacks and queues:** Arrays are used as the underlying data structure for implementing stacks and queues, which are commonly used in algorithms and data structures.
* **Graphs**: Arrays can be used to represent graphs in computer science. Each element in the array represents a node in the graph, and the relationships between the nodes are represented by the values stored in the array.
* **Dynamic programming**: Dynamic programming algorithms often use arrays to store intermediate results of subproblems in order to solve a larger problem.

Below are some real-time applications of arrays.

* **Signal Processing:** Arrays are used in signal processing to represent a set of samples that are collected over time. This can be used in applications such as speech recognition, image processing, and radar systems.
* **Multimedia Applications:** Arrays are used in multimedia applications such as video and audio processing, where they are used to store the pixel or audio samples. For example, an array can be used to store the RGB values of an image.
* **Data Mining:** Arrays are used in data mining applications to represent large datasets. This allows for efficient data access and processing, which is important in real-time applications.
* **Robotics**: Arrays are used in robotics to represent the position and orientation of objects in 3D space. This can be used in applications such as motion planning and object recognition.
* **Real-time Monitoring and Control Systems:** Arrays are used in real-time monitoring and control systems to store sensor data and control signals. This allows for real-time processing and decision-making, which is important in applications such as industrial automation and aerospace systems.
* **Financial Analysis:** Arrays are used in financial analysis to store historical stock prices and other financial data. This allows for efficient data access and analysis, which is important in real-time trading systems.
* **Scientific Computing:**Arrays are used in scientific computing to represent numerical data, such as measurements from experiments and simulations. This allows for efficient data processing and visualization, which is important in real-time scientific analysis and experimentation.

**Advantages of array data structure:**

* **Efficient access to elements:** Arrays provide direct and efficient access to any element in the collection. Accessing an element in an array is an O(1) operation, meaning that the time required to access an element is constant and does not depend on the size of the array.
* **Fast data retrieval:**Arrays allow for fast data retrieval because the data is stored in contiguous memory locations. This means that the data can be accessed quickly and efficiently without the need for complex data structures or algorithms.
* **Memory efficiency:** Arrays are a memory-efficient way of storing data. Because the elements of an array are stored in contiguous memory locations, the size of the array is known at compile time. This means that memory can be allocated for the entire array in one block, reducing memory fragmentation.
* **Versatility:** Arrays can be used to store a wide range of data types, including integers, floating-point numbers, characters, and even complex data structures such as objects and pointers.
* **Easy to implement:**Arrays are easy to implement and understand, making them an ideal choice for beginners learning computer programming.
* **Compatibility with hardware:**The array data structure is compatible with most hardware architectures, making it a versatile tool for programming in a wide range of environments.

**Disadvantages of array data structure:**

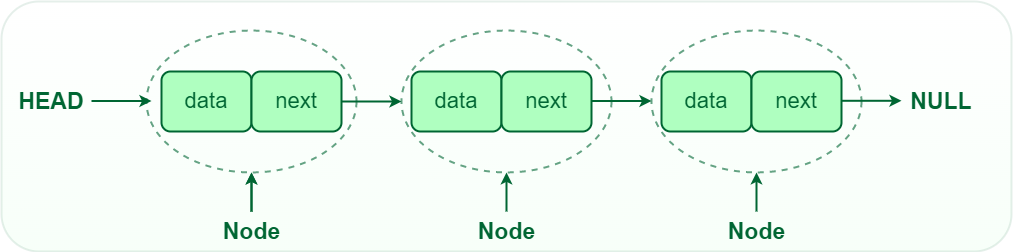
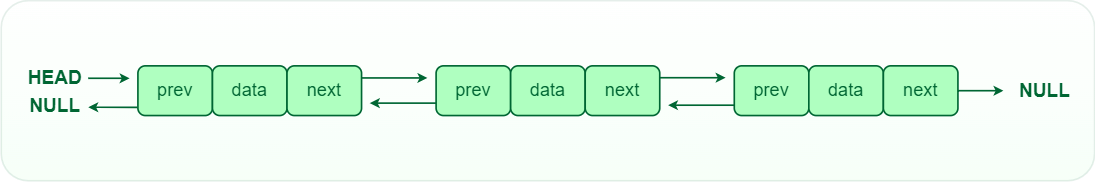
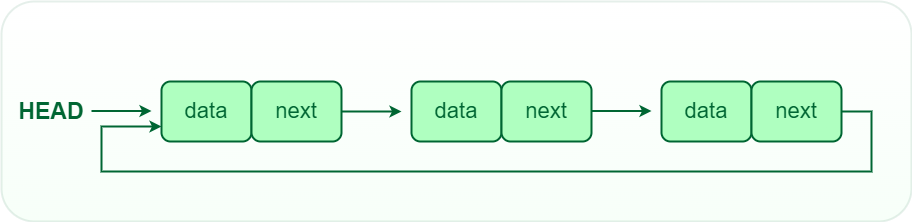
* **Fixed size:** Arrays have a fixed size that is determined at the time of creation. This means that if the size of the array needs to be increased, a new array must be created and the data must be copied from the old array to the new array, which can be time-consuming and memory-intensive.
* **Memory allocation issues:** Allocating a large array can be problematic, particularly in systems with limited memory. If the size of the array is too large, the system may run out of memory, which can cause the program to crash.
* **Insertion and deletion issues**: Inserting or deleting an element from an array can be inefficient and time-consuming because all the elements after the insertion or deletion point must be shifted to accommodate the change.
* **Wasted space:** If an array is not fully populated, there can be wasted space in the memory allocated for the array. This can be a concern if memory is limited.
* **Limited data type support:**Arrays have limited support for complex data types such as objects and structures, as the elements of an array must all be of the same data type.
* **Lack of flexibility:**The fixed size and limited support for complex data types can make arrays inflexible compared to other data structures such as linked lists and trees.

*A Linked List is a****linear data structure****which looks like a chain of nodes, where each node is a different element. Unlike Arrays, Linked List elements are not stored at a contiguous location.*

It is basically**chains of nodes**, each node contains information such as **data** and a **pointer to the next node** in the chain. In the linked list there is a **head pointer**, which points to the first element of the linked list, and if the list is empty then it simply points to null or nothing.

**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.
* [**1. Singly-linked list**](https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/)
* Traversal of items can be done in the forward direction only due to the linking of every node to its next node.
* [](https://media.geeksforgeeks.org/wp-content/uploads/20220712172013/Singlelinkedlist.png)
* *Singly Linked List*
* [**2. Doubly linked list**](https://www.geeksforgeeks.org/doubly-linked-list/)
* Traversal of items can be done in both forward and backward directions as every node contains an additional **prev** pointer that points to the previous node.
* [](https://media.geeksforgeeks.org/wp-content/uploads/20220712180755/Doublylinkedlist.png)
* *Doubly linked list*
* [**3. Circular linked lists**](https://www.geeksforgeeks.org/circular-linked-list/)
* A circular linked list is a type of linked list in which the first and the last nodes are also connected to each other to form a circle, there is no NULL at the end.
* [](https://media.geeksforgeeks.org/wp-content/uploads/20220712181336/Circularlinkedlist.png)
* *Circular Linked List*

## [Disadvantages of Linked Lists:](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-linked-list/#:~:text=Advantages%20of%20Linked%20Lists%3A)

* **Memory usage:**The use of pointers is more in linked lists hence, complex and requires more memory.
* **Accessing a node:**Random access is not possible due to dynamic memory allocation.
* **Search operation costly:**Searching for an element is costly and requires O(n) time complexity.
* **Traversing in reverse order:**Traversing is more time-consuming and reverse traversing is not possible in singly linked lists.

## [****Applications of Linked List:****](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-linked-list/#:~:text=Applications%20of%20Linked%20Lists%3A)

Here are some of the applications of a linked list:

* Linear data structures such as stack, queue, and non-linear data structures such as hash maps, and graphs can be implemented using linked lists.
* **Dynamic memory allocation:** We use a linked list of free blocks.
* **Implementation of graphs:**Adjacency list representation of graphs is the most popular in that it uses linked lists to store adjacent vertices.
* In web browsers and editors, doubly linked lists can be used to build a forwards and backward navigation button.
* A circular doubly linked list can also be used for implementing data structures like Fibonacci heaps.

## [Applications of Linked Lists in real world:](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-linked-list/#:~:text=Applications%20of%20Linked%20Lists%3A)

* The list of songs in the music player is linked to the previous and next songs.
* In a web browser, previous and next web page URLs are linked through the previous and next buttons.
* In the image viewer, the previous and next images are linked with the help of the previous and next buttons.
* Switching between two applications is carried out by using **“alt+tab**” in windows and “**cmd+tab**” in mac book. It requires the functionality of a circular linked list.
* In mobile phones, we save the contacts of people. The newly entered contact details will be placed at the correct alphabetical order.
* This can be achieved by a linked list to set contact at the correct alphabetical position.
* The modifications that we made in the documents are actually created as nodes in doubly linked list. We can simply use the undo option by pressing **Ctrl+Z** to modify the contents. It is done by the functionality of a linked list.

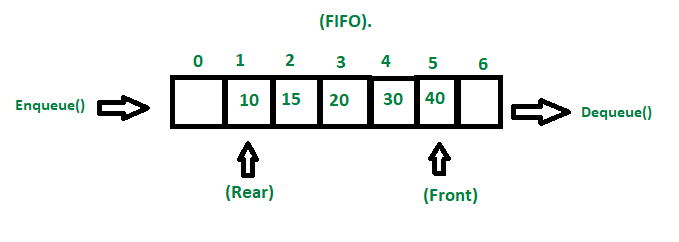
## ****What is Queue?****

*A****queue****is a*[*linear data structure*](https://www.geeksforgeeks.org/introduction-to-linear-data-structures/)*that is open at both ends and the operations are performed in*[*First In First Out (FIFO)*](https://www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-programming/)*order.*

*We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end.  The element which is first pushed into the order, the delete operation is first performed on that.*

## [****FIFO Principle of Queue****](https://www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-programming/)****:****

* A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).
* Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the **front** of the queue(sometimes, **head** of the queue), similarly, the position of the last entry in the queue, that is, the one most recently added, is called the **rear** (or the**tail**) of the queue. See the below figure.



*FIFO property of queue*

## ****Characteristics of Queue:****

* Queue can handle multiple data.
* We can access both ends.
* They are fast and flexible.

## [Types of Queue:](https://www.geeksforgeeks.org/different-types-of-queues-and-its-applications/)

There are different types of queues:

1. **Input Restricted Queue:** This is a simple queue. In this type of queue, the input can be taken from only one end but deletion can be done from any of the ends.
2. **Output Restricted Queue:** This is also a simple queue. In this type of queue, the input can be taken from both ends but deletion can be done from only one end.
3. [**Circular Queue**](https://www.geeksforgeeks.org/introduction-and-array-implementation-of-circular-queue/)**:** This is a special type of queue where the last position is connected back to the first position. Here also the operations are performed in FIFO order. To know more refer [this](https://www.geeksforgeeks.org/introduction-and-array-implementation-of-circular-queue/).
4. [**Double-Ended Queue (Dequeue)**](https://www.geeksforgeeks.org/deque-set-1-introduction-applications/)**:** In a double-ended queue the insertion and deletion operations, both can be performed from both ends. To know more refer [this](https://www.geeksforgeeks.org/deque-set-1-introduction-applications/).
5. [**Priority Queue**](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/)**:** A priority queue is a special queue where the elements are accessed based on the priority assigned to them. To know more refer [this](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/).

**Basic Operations for Queue in Data Structure:**

Some of the basic operations for Queue in Data Structure are:

1. **Enqueue() –** Adds (or stores) an element to the end of the queue..
2. **Dequeue() –** Removal of elements from the queue.
3. **Peek() or front()-** Acquires the data element available at the front node of the queue without deleting it.
4. **rear() –** This operation returns the element at the rear end without removing it.
5. **isFull() –** Validates if the queue is full.
6. **isNull() –** Checks if the queue is empty.

[**Applications of Queue:**](https://www.geeksforgeeks.org/applications-of-queue-data-structure/)

Application of queue is common. In a computer system, there may be queues of tasks waiting for the printer, for access to disk storage, or even in a time-sharing system, for use of the CPU. Within a single program, there may be multiple requests to be kept in a queue, or one task may create other tasks, which must be done in turn by keeping them in a queue.

* It has a single resource and multiple consumers.
* It synchronizes between slow and fast devices.
* In a network, a queue is used in devices such as a router/switch and mail queue.
* Variations: dequeue, priority queue and double-ended priority queue.