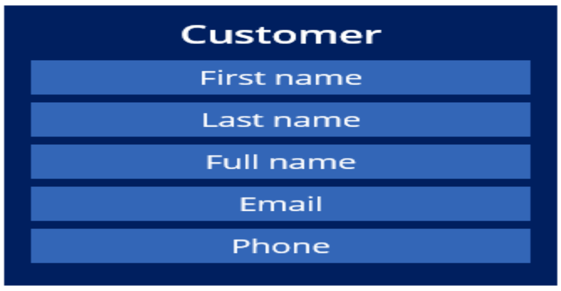
UNIT-1

Data: Data is defined as the collection of raw materials

Data Object: A data object is a structure for describing a data entity by grouping a set of related fields.

Example: supermarket Online orders application might contain a Customer data object. As seen in the following image, the Customer data object includes fields that describe the supermarket's customer, such as **First name, Last name, Full name, Email, and Phone.**



Data objects hold data for your application and simplify the organization of fields, user interface views, and integration settings that your application needs to access the right data at the right time to successfully resolve a Case.

What is a data type?

In software programming, data type refers to the type of value a variable has and what type of mathematical, relational or logical operations can be applied without causing an error.

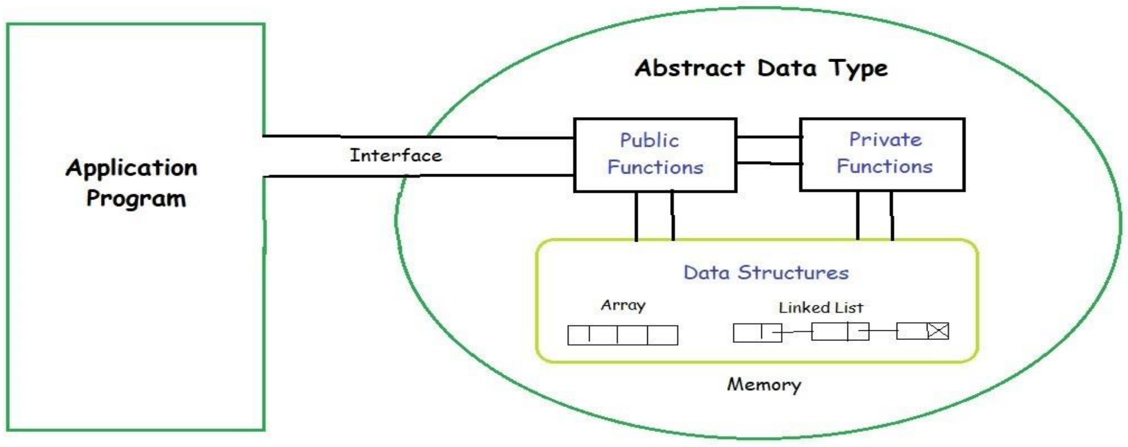
Data Type	Data Structure
The data type is the form of a variable to which a value can be assigned. It defines that the particular variable will assign the values of the given data type only.	Data structure is a collection of different kinds of data. That entire data can be represented using an object and can be used throughout the program.
It can hold value but not data. Therefore, it is data less.	It can hold multiple types of data within a single object.
The implementation of a data type is known as abstract implementation.	Data structure implementation is known as concrete implementation.
There is no time complexity in the case of data types.	In data structure objects, time complexity plays an important role.
Data type examples are int, float, double, etc.	Data structure examples are stack, queue, tree, etc.

Abstract Data Type and Data Structures

An Abstract Data Type (ADT) is a programming concept that defines a high-level view of a data structure, without specifying the implementation details. In other words, it is a blueprint for creating a data structure that defines the behavior and interface of the structure, without specifying how it is implemented.

An ADT in the data structure can be thought of as a set of operations that can be performed on a set of values. This set of operations actually defines the behavior of the data structure, and they are used to manipulate the data in a way that suits the needs of the program.

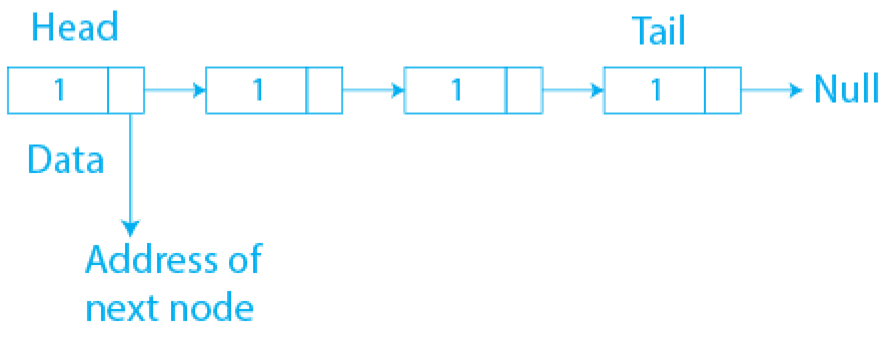
Abstract Data Type Model



Examples of abstract data type in data structures are List, Stack, Queue, etc.

List ADT

Lists are linear data structures that hold data in a non-continuous structure. The list is made up of data storage containers known as "nodes." These nodes are linked to one another, which means that each node contains the address of another block. All of the nodes are thus connected to one another via these links.

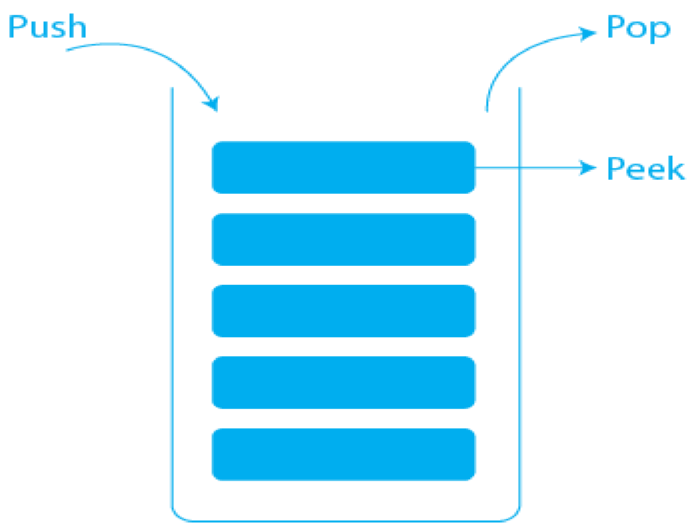


Some of the most essential operations defined in List ADT are listed below.

- front(): returns the value of the node present at the front of the list.
- back(): returns the value of the node present at the back of the list.
- push_front(int val): creates a pointer with value = val and keeps this pointer to the front of the linked list.
- push_back(int val): creates a pointer with value = val and keeps this pointer to the back of the linked list.
- pop_front(): removes the front node from the list.
- pop_back(): removes the last node from the list.
- empty(): returns true if the list is empty, otherwise returns false.
- size(): returns the number of nodes that are present in the list.

Stack ADT

A stack is a linear data structure that only allows data to be accessed from the top. It simply has two operations: push (to insert data to the top of the stack) and pop (to remove data from the stack). (used to remove data from the stack top).

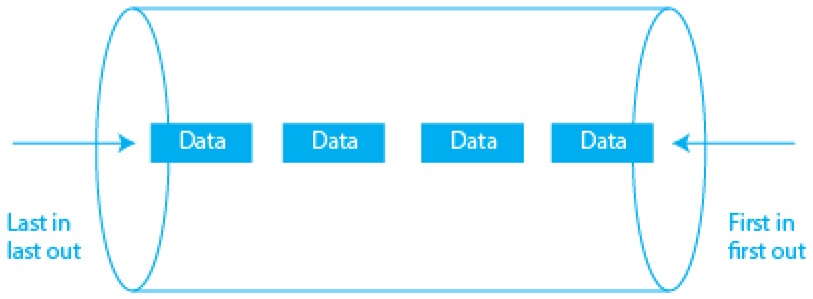


Some of the most essential operations defined in Stack ADT are listed below.

- top(): returns the value of the node present at the top of the stack.
- push(int val): creates a node with value = val and puts it at the stack top.
- pop(): removes the node from the top of the stack.
- empty(): returns true if the stack is empty, otherwise returns false.
- size(): returns the number of nodes that are present in the stack.

Queue ADT

A queue is a linear data structure that allows data to be accessed from both ends. There are two main operations in the queue: push (this operation inserts data to the back of the queue) and pop (this operation is used to remove data from the front of the queue).



Some of the most essential operations defined in Queue ADT are listed below.

- front(): returns the value of the node present at the front of the queue.
- back(): returns the value of the node present at the back of the queue.
- push(int val): creates a node with value = val and puts it at the front of the queue.
- pop(): removes the node from the rear of the queue.
- empty(): returns true if the queue is empty, otherwise returns false.
- size(): returns the number of nodes that are present in the queue.

Advantages of ADT in Data Structures

The advantages of ADT in Data Structures are:

- Provides abstraction, which simplifies the complexity of the data structure and allows users to focus on the functionality.
- Enhances program modularity by allowing the data structure implementation to be separate from the rest of the program.
- Enables code reusability as the same data structure can be used in multiple programs with the same interface.
- Promotes the concept of data hiding by encapsulating data and operations into a single unit, which enhances security and control over the data.

Concepts of static and dynamic

What is a Static Data structure?

In Static data structure the size of the structure is fixed. The content of the data structure can be modified but without changing the memory space allocated to it.

Example of Static Data Structures: **Array**

What is Dynamic Data Structure?

In Dynamic data structure the size of the structure is not fixed and can be modified during the operations performed on it. Dynamic data structures are designed to facilitate change of data structures in the run time.

Example of Dynamic Data Structures: **Linked List**

Aspect	Static Data Structure	Dynamic Data Structure
Memory allocation	Memory is allocated at compile-time	Memory is allocated at run-time
Size	Size is fixed and cannot be modified	Size can be modified during runtime
Memory utilization	Memory utilization may be inefficient	Memory utilization is efficient as memory can be reused
Access	Access time is faster as it is fixed	Access time may be slower due to indexing and pointer usage
Examples	Stacks, Queues, Trees (with fixed size)	Lists, Trees (with variable size), Hash tables

Advantage of Static data structure:

Fast access time: Static data structures offer fast access time because memory is allocated at compile-time and the size is fixed, which makes accessing elements a simple indexing operation.

Predictable memory usage: Since the memory allocation is fixed at compile-time, the programmer can precisely predict how much memory will be used by the program, which is an important factor in memory-constrained environments.

Ease of implementation and optimization: Static data structures may be easier to implement and optimize since the structure and size are fixed, and algorithms can be optimized for the specific data structure, which reduces cache misses and can increase the overall performance of the program.

Efficient memory management: Static data structures allow for efficient memory allocation and management. Since the size of the data structure is fixed at compile-time, memory can be allocated and released efficiently, without the need for frequent reallocations or memory copies.

Simplified code: Since static data structures have a fixed size, they can simplify code by removing the need for dynamic memory allocation and associated error checking.

Advantage Of Dynamic Data Structure :

Flexibility: Dynamic data structures can grow or shrink at runtime as needed, allowing them to adapt to changing data requirements. This flexibility makes them well-suited for situations where the size of the data is not known in advance or is likely to change over time.

Reduced memory waste: Since dynamic data structures can resize themselves, they can help reduce memory waste. For example, if a dynamic array needs to grow, it can allocate additional memory on the heap rather than reserving a large fixed amount of memory that might not be used.

Improved performance for some operations: Dynamic data structures can be more efficient than static data structures for certain operations. For example, inserting or deleting elements in the middle of a dynamic list can be faster than with a static array, since the remaining elements can be shifted over more efficiently.

Simplified code: Dynamic data structures can simplify code by removing the need for manual memory management. Dynamic data structures can also reduce the complexity of code for data structures that need to be resized frequently.

Scalability: Dynamic data structures can be more scalable than static data structures, as they can adapt to changing data requirements as the data grows.

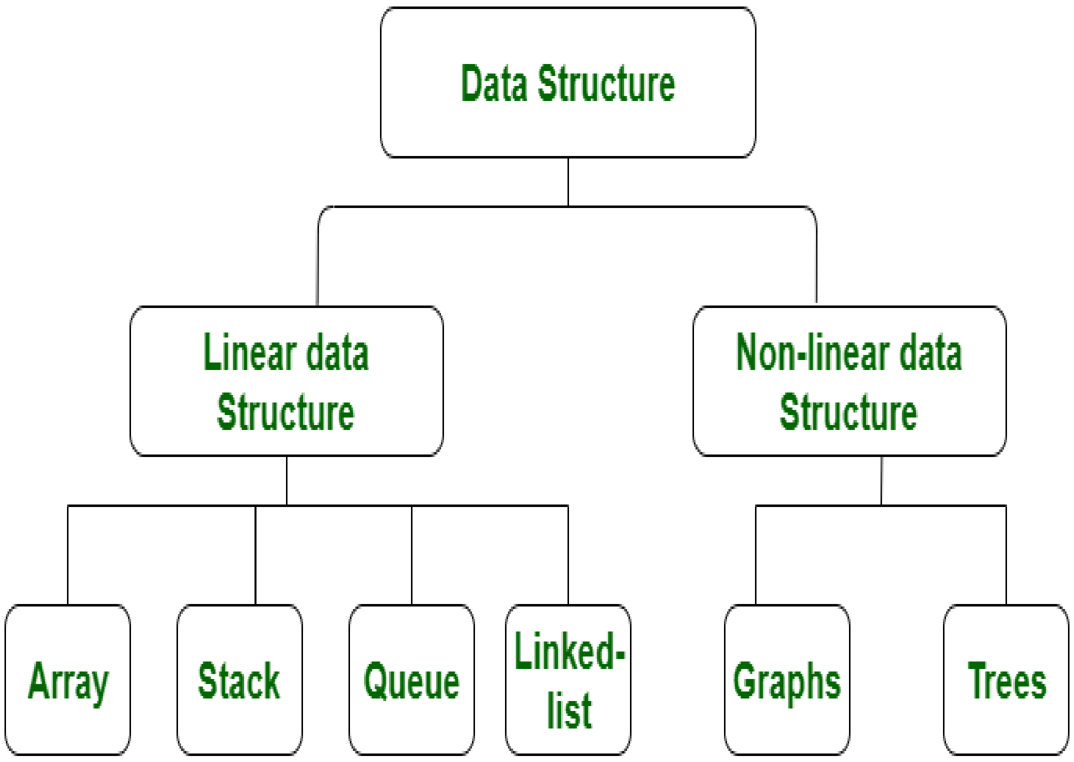
Linear and Nonlinear Data Structures:

Linear Data Structure:

- Data structure where data elements are arranged sequentially or linearly where each and every element is attached to its previous and next adjacent is called a linear data structure.
- In linear data structure, single level is involved. Therefore, we can traverse all the elements in single run only. Linear data structures are easy to implement because computer memory is arranged in a linear way.
- Its examples are array, stack, queue, linked list, etc.

Non-linear Data Structure:

- Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures.
- In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only. Non-linear data structures are not easy to implement in comparison to linear data structure. It utilizes computer memory efficiently in comparison to a linear data structure.
- Its examples are trees and graphs.


Data Structure
Linear data Structure
Non-linear data Structure
Array
Stack
Queue
Linked-list
Graphs
Trees