**DATA STRUCTURES USING C**

Let us begin with the very basic definitions of “What actually the DataStructure mean?”and proceed further with the detailed information…

**What is Data Structure?**

Data Structure is a way to store and organize data so that it can be used efficiently. The data structure name itself indicates that organizing the data in memory. There are many ways of organizing the data in the memory.

**Types of Data Structures**

There are two types of data structures:

* Primitive data structure
* Non-primitive data structure

**Data structures can also be classified as:**

**Static data structure:** It is a type of data structure where the size is allocated at the compile time. Therefore, the maximum size is fixed.

**Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible.

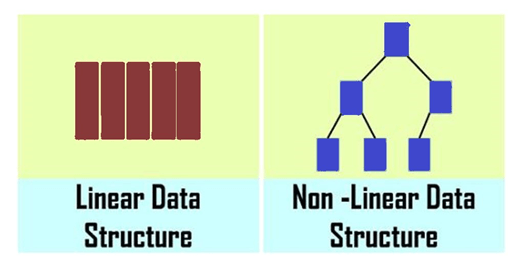
**Primitive Data structure**

The primitive data structures are primitive data types. The int, char, float, double, and pointer are the primitive data structures that can hold a single value.

**Non-Primitive Data structure**

The non-primitive data structure is divided into two types:

* Linear data structure
* Non-linear data structure



**Linear Data Structure**

The arrangement of data in a sequential manner is known as a linear data structure.

The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues.

In these data structures, one element is connected to only one another element in a linear form.

## Non-linear Data Structure

## The data structure where data items are not organized sequentially is called [a non-linear data structure](https://theknowshares.com/computerscience/datastructure/nonlinear-data-structures/).

**When one element is connected to the 'n' number of elements known as a non-linear data structure.**

**The best example is trees and graphs. In this case, the elements are arranged in a random manner.**

**Advantages of Data structures**

**The following are the advantages of a data structure:**

* **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
* **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.
* **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

Let us see in detail about one of the linear Datastructure-Stack.

**Stack:**

Stack is one of the important linear data structures based on the LIFO ( Last In First Out ) principle. Many computer applications and the various strategies used in the operating system and other places are based on the principle of LIFO itself. In this principle, the data element entered last must be popped out first from it, and the element pushed into the stack at the very first time is popped out last. In this approach, we will push the data elements into the stack until it reaches their end limit; after that, we will pop out the corresponding values.

In other words*, a*stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

**Real-world Analog:**stack of plates, where the last plate placed on top is the first one to be removed.

**Basic Operations**

**Push**: Adding an element to the top of the stack.

**Pop**: Removing the top element from the stack.

**Peek/Top**: Viewing the top element without removing it.

**isEmpty**: Checking if the stack is empty.

**isFull**: Checking if the stack is full (applicable in array-based implementation).

**Implementation Methods**

**Array-Based Implementation**

**Pros**: Simple and fast, as accessing elements by index is O(1).

**Cons**: Fixed size, which limits the number of elements.

**Code Example**: Basic structure and operations using arrays.

**Linked List-Based Implementation**

**Pros**: Dynamic size, no fixed limit.

**Cons**: Slightly more complex and involves pointer manipulation.

**Code Example**: Basic structure and operations using linked lists.

### Working of Stack

Suppose we want to store the elements in a stack and let's assume that stack is empty. We have taken the stack of size 5 as shown below in which we are pushing the elements one by one until the stack becomes full.

### DS Stack Introduction

Since our stack is full as the size of the stack is 5. In the above cases, we can observe that it goes from the top to the bottom when we were entering the new element in the stack. The stack gets filled up from the bottom to the top.

When we perform the delete operation on the stack, there is only one way for entry and exit as the other end is closed. It follows the LIFO pattern, which means that the value entered first will be removed last. In the above case, the value 5 is entered first, so it will be removed only after the deletion of all the other elements.

**Applications of Stack**

The following are the applications of the stack:

**Balancing of symbols:** Stack is used for balancing a symbol.

**String reversal:** Stack is also used for reversing a string.

**UNDO/REDO:** It can also be used for performing UNDO/REDO operations.

**Recursion:** The recursion means that the function is calling itself again.

**DFS(Depth First Search):** This search is implemented on a Graph, and Graph uses the stack data structure.

**Backtracking:** Suppose we have to create a path to solve a maze problem. If we are moving in a particular path, and we realize that we come on the wrong way. In order to come at the beginning of the path to create a new path, we have to use the stack data structure.

**Expression conversion:** Stack can also be used for expression conversion. This is one of the most important applications of stack. The list of the expression conversion is given below:

* Infix to prefix
* Infix to postfix
* Prefix to infix
* Prefix to postfix

Postfix to infix

**Memory management:** The stack manages the memory. The memory is assigned in the contiguous memory blocks. The memory is known as stack memory as all the variables are assigned in a function call stack memory. The memory size assigned to the program is known to the compiler. When the function is created, all its variables are assigned in the stack memory. When the function completed its execution, all the variables assigned in the stack are released.

### ****Example Code for Array-Based Stack in C****

#include <stdio.h>

#define MAX 100

int stack[MAX];

int top = -1;

void push(int data) {

if (top == MAX - 1) {

printf("Stack overflow\n");

return;

}

stack[++top] = data;

}

int pop() {

if (top == -1) {

printf("Stack underflow\n");

return -1;

}

return stack[top--];

}

int peek() {

if (top == -1) {

printf("Stack is empty\n");

return -1;

}

return stack[top];

}

int isEmpty() {

return top == -1;

}

int isFull() {

return top == MAX - 1;

}

int main() {

push(10);

push(20);

printf("Top element: %d\n", peek());

printf("Popped element: %d\n", pop());

printf("Top element after pop: %d\n", peek());

return 0;

### }

### ****Example Code for Linked List-Based Stack in C****

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* top = NULL;

void push(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (!newNode) {

printf("Stack overflow\n");

return;

}

newNode->data = data;

newNode->next = top;

top = newNode;

}

int pop() {

if (!top) {

printf("Stack underflow\n");

return -1;

}

Node\* temp = top;

int data = top->data;

top = top->next;

free(temp);

return data;

}

int peek() {

if (!top) {

printf("Stack is empty\n");

return -1;

}

return top->data;

}

int isEmpty() {

return top == NULL;

}

int main() {

push(10);

push(20);

printf("Top element: %d\n", peek());

printf("Popped element: %d\n", pop());

printf("Top element after pop: %d\n", peek());

return 0;

### }

**Advantages**:

Simple to implement and use.

Efficient for certain types of algorithms like depth-first search.

**Disadvantages**:

Limited by size in array implementation.

Can lead to stack overflow if not managed properly in recursion.