COMPUTER SCIENCE

B. Sc. I (CBCS)
Semester-II
2023-2024

1CS2 : Data Structure & OOPS

Unit-I : Introduction to Data Structure

PROF. V. V. AGARKAR
Assistant Professor & Head
Department of Computer Science

Shri. D. M. Burungale Science & Arts College, Shegaon, Dist. Buldana

UNIT – I

DATA STRUCTURE

Syllabus: **Data structure:** Introduction to data structure, types of data structure: Primitive and Non-primitive, Linear and Non-linear data structure, Data structure operations. **Arrays:** Definition and concepts, Memory representations, Operations: Traversing, Insertion, Deletion. **Stacks:** Definition and concepts, Memory representations, Operations: Traversing, Insertion, Deletion.

Introduction

A *data structure* is an arrangement of data in a computer's memory. A data structure is a way of organizing data that considers not only the items stored, but also their relationship to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data. A data structure not only defines what elements it may contain, it also supports a set of operations on these elements.

```
Data structure = organized data + operations
```

The data structure deals with the study of how the data is organized. Data structure is the most convenient way to handle data of different data types for a known problem.

Areas in which data structures are applied (Applications)

- Compiler Design
- Operating System
- Database Management System
- Statistical analysis package
- Numerical Analysis,
- Graphics,
- Artificial Intelligence,
- Simulation

Type of Data Structures

The data structures can be classified into two different types namely

1) Primitive data structures, and
2) Non - primitive data structures

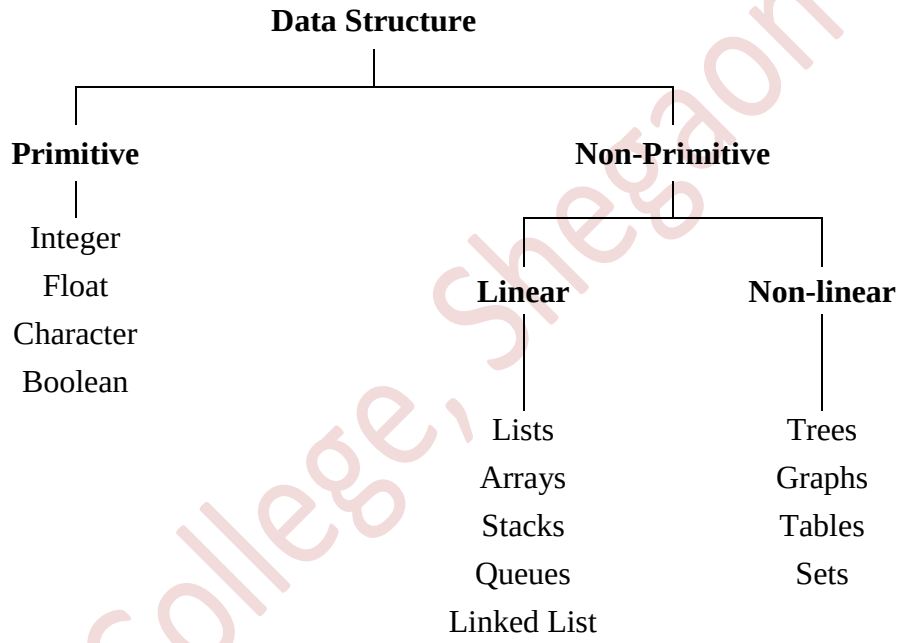
1) ***Primitive data structures***

The data structures which can be directly operated are called primitive data structures. The integer, float, character etc. are some of the primitive data structures. Operations like deletion, selection and updation can be carried out on primitive data structures.

2) ***Non - primitive data structures***

The data structures which are not primitive means which cannot be manipulated directly, instead they are derived from primitive data structures are called non-primitive data structures. Non-primitive data structures can be again classified into two different types namely (i) Linear data structure and (ii) Non-Linear data structure.

1) **Linear Data Structure:** A Data structure where all data elements are arranged sequentially or linearly and each member element is connected to its previous and next adjacent element. Here a single level is involved and hence can be traversed in a single run. These are easy to implement because computer memory is also linear. Examples of linear data structures are array, stack, queue, linked list, etc.
2) **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 rather multiple levels are involved. Therefore, all the elements can't be traversed in single run only. These are not easy to implement. Examples of non-linear data structures are trees and graphs.



[Fig. 1 : Classification of Data Structure **]**

Data structure operations

The data can be processed by means of certain operations. Following are some operations with data structures:

1. **Traversing:** The traversing operation allows a user to access or visit each element within data structure exactly once so that certain items in the element may be processed.
2. **Searching:** The searching operation allows finding out the location of the element using a given key value within a data structure to check whether the element is present or not.
3. **Insertion:** The insertion operation allows adding a new element into a data structure at the specified position.
4. **Deletion:** The deletion operation allows removing an element from the data structure.
5. **Sorting:** The sorting operation allows arranging the elements in some logical order (alphabetically, ascending or descending order).

6. **Merging:** The merging operation allows combining the elements from two or more sorted data structures into a single sorted data structure.
7. **Copying:** The copying operation allows copying the contents from a data structure to another data structure.
8. **Concatenation:** The concatenation operation allows joining the elements of one data followed by another data structure into a new resultant data structure.

Linear Arrays (or Arrays)

An *array* or *linear array* is a finite, ordered collection of homogeneous data elements. Array is finite because it contains only limited number of elements; and ordered, as all the elements are stored one by one in contiguous locations of computer memory in a linear ordered fashion. All the elements of an array are of the same data type (say, integer) only and hence it is termed as collection of homogeneous elements.

If we choose the name ***A*** for the array, then the elements of ***A*** are denoted by ***subscript*** notation, as follows:

$A_1, A_2, A_3, \ldots, A_n$

or by the parenthesis notation

```
A(1), A(2), A(3), ..., A(N)
```

or by the bracket notation

```
A[1],  A[2], A[3], ..., A[N]
```

Regardless of the notation, the number K in A[K] is called a *subscript* and A[K] is called a *subscripted variable*.

Example:

Let DATA be a 5-element linear array of integers such that

```
DATA[1]=247   DATA[2]=56   DATA[3]=429   DATA[4]=135   DATA[5]=87
```

Sometimes such an array denotes by simply writing

```
DATA:      247,  56,  429,  135,  87
```

The array DATA is frequently pictured as in Fig. 2(*a*) or Fig. 2(*b*).

DATA

1	247
2	56
3	429
4	135
5	87

(a)

DATA

247	56	429	135	87
1	2	3	4	5

(b)

[Fig. 2 : Array **]**

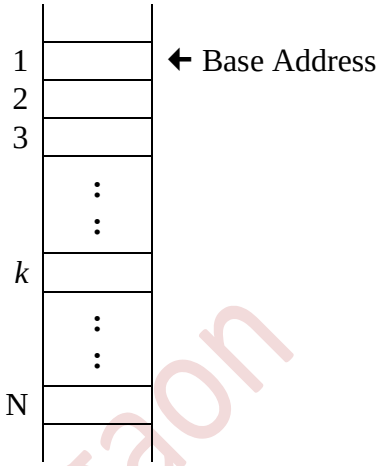
Representation of Linear Arrays in memory

The memory of computer is simply a sequence of addressed locations as shown in Fig. 3. Let LA be a linear array stored in the memory of computer.

As the elements of the array LA are stored in the consecutive memory cells, the computer does not need to keep track of the address of every element of the array. It only needs to keep track of the address of the first element of the array which is called *Base Address* of an array. Using this base address, the computer calculates the address of any element of the array by using the following formula:

```
Address(LA[k]) =  Base Address +
                  w(k - LB)
```

Where *w* is the size of the data type of the array LA and LB is lower bound of the array.



[Fig. 3: Memory representation of Array **]**

Traversing linear arrays

Traversing means accessing and processing (frequently called ***visiting***) each element of array exactly once. The following algorithm traverses a linear array LA.

Algorithm

```
Traverse(LA, LB, UB, PROCESS)
Here LA is a linear array with lower
bound LB and upper bound UB. This
algorithm traverses LA applying an
operation PROCESS to each element of LA.

1. [Initialize counter]
      Set K = LB
2. Repeat Steps 3 and 4 while K <= UB
3.     [Visit element]
       Apply PROCESS to LA[K]
4.     [Increase counter]
       Set K = K + 1
   [End of Step 2 loop]
5. Exit
```

Inserting into a Linear Array

Inserting refers to the operation of adding one new element to the array. Inserting an element at the "end" of an array can be easily done, but, to insert an element in the middle (or at the beginning), some of the elements must be moved downward to new locations to accommodate the new element and keep the order of the elements.

The following algorithm inserts a data element ITEM into the K^{th} position in a linear array LA with N elements. The first four steps create space in LA by moving downward one location each element from the K^{th} position on. These elements are moved in reverse order—i.e. first LA[N], then LA[N-1], ... and last LA[K]; otherwise data might be erased.

Algorithm

```
INSERT(LA, N, K, ITEM)

Here LA is a linear array with N elements
and K is a positive integer such that K <=
N. This algorithm inserts an element ITEM
into the Kth position in LA.

1. [Initialize counter]
   Set J = N

2. Repeat Steps 3 and 4 while J >= K

3.     [Move Jth element downward]
       Set LA[J+1] = LA[J]

4.     [Decrease counter]
       Set J = J-1
   [End of Step 2 loop]

5. [Insert element]
   Set LA[K] = ITEM

6. [Reset N]
   Set N = N + 1

7. Exit.
```

Deleting from a Linear Array

Deleting refers to the operation of removing one of the elements from array. Deleting an element at the "end" of an array presents no difficulties, but deleting an element somewhere in the middle of the array would require that each subsequent element be moved one location upward in order to "fill up" the array.

The following algorithm deletes the K^{th} element from a linear array LA and assigns it to a variable ITEM.

Algorithm

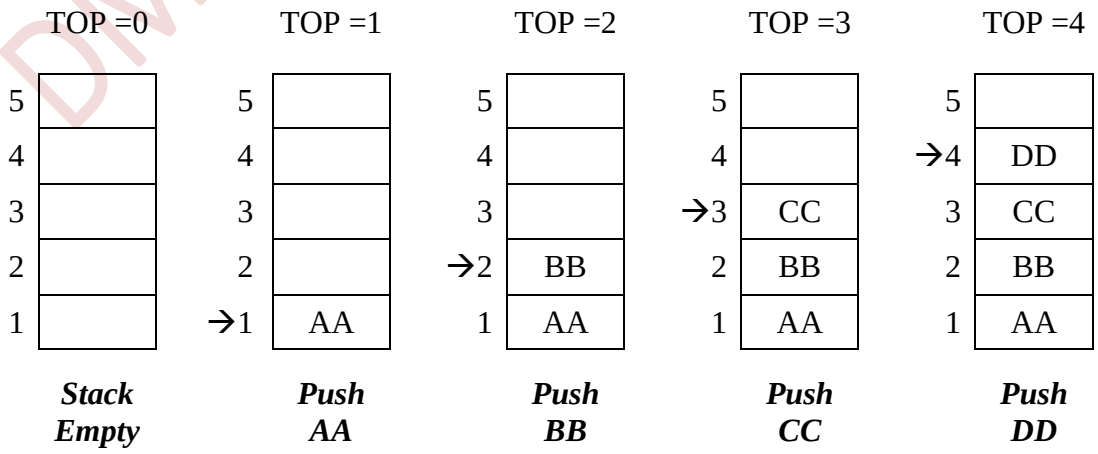
```
DELETE(LA, N, K, ITEM)

Here LA is a linear array with N elements and
K is a positive integer such that K <= N. This
algorithm deletes the K^th element from LA.

1. Set ITEM = LA[K]

2. Repeat for J = K to N – 1:
        [Move J+1^st element upward]
         Set LA[J] = LA[J+1]
   [End of loop]

3. [Reset the number N of elements in LA]
   Set N = N – 1

4. Exit.
```

Stack

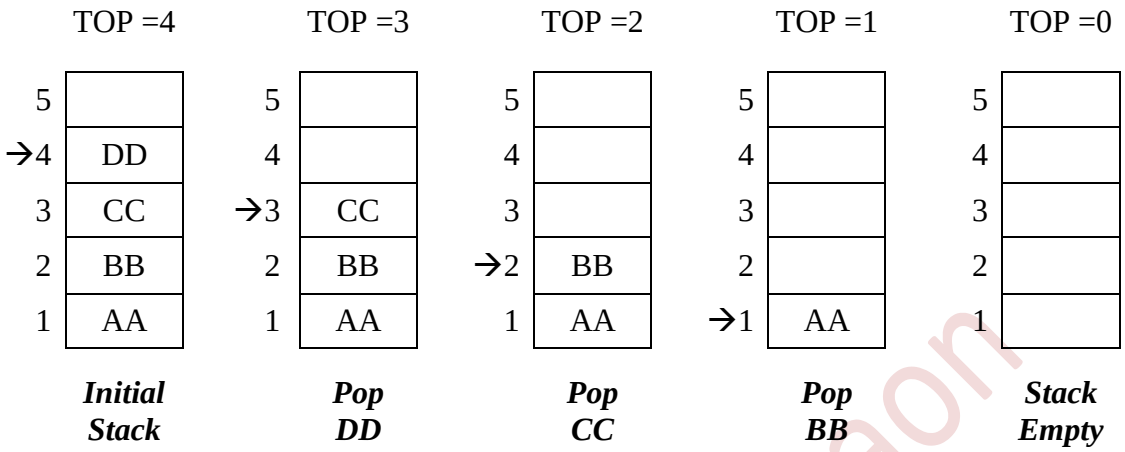
A stack is a linear structure in which items may be inserted or removed only at one end (same end) called the ***top*** of the stack. That means it is possible to remove elements from a stack in reverse order from the insertion of elements into the stack. Thus, a stack data structure has the **LIFO** (***Last In First Out***) property.

Generally, two operations are associated with the stacks named Push & Pop.

- *Push* is an operation used to insert an element at the top.
- *Pop* is an operation used to delete an element from the top



[Fig. 4: *PUSH operations on to Stack.***]**



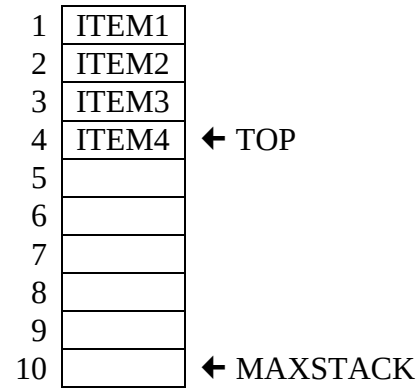
[Fig. 5: *POP operations on to Stack.***]**

Representation of stack in the Computer's memory

Stacks may be represented in the computer in various ways, usually by means of a linear array. Here stack is represented by linear array **STACK**, TOP is a pointer variable which contains the location of the top element of STACK. Initially it is assign to 0 to show that stack is empty and **MAXSTACK** represents the maximum possible number of elements in the stack.

When there is no element in the stack to remove then this is known as ***stack underflow***. When the stack contains equal number of elements as per its capacity and no more elements can be added such status of stack is known as ***stack overflow***.

Fig. 6 shows array representation of stack. The TOP is pointing to 4 which says that stack has four items and as the MAXSTACK = 10, there is still space for accommodating six elements.



[Fig. 6 : Memory representation of Stack **]**

Applications of Stack

Various applications of stack are:

1. Expression conversion.
2. Expression Evaluation.
3. Parsing well formed parenthesis.
4. Decimal to binary conversion.
5. Reversing a string.
6. Storing function calls.

Adding (pushing) an item onto the stack

To add an item on to stack PUSH operation is performed. While executing procedure PUSH, first test whether there is a room in the stack for the new item; if not, then the condition *Overflow* occurs. Following is algorithm to push an item on to stack:

Algorithm

```
PUSH(STACK, TOP, MAXSTACK, ITEM)
Step 1:   [Check for stack overflow]
          If TOP = MAXSTACK
          Then print "Stack overflow" and exit.
Step 2:   [Increment the TOP value by one]
          TOP = TOP + 1
Step 3:   [Insert the ITEM in new TOP position]
          STACK[TOP] = ITEM
Step 4:   Exit.
```

Deleting (popping) an item from the stack

To delete an item from stack POP operation is performed. While executing procedure POP, first test whether there is an element in the stack to be deleted; if not, then the condition *Underflow* occurs. Following is algorithm to pop an item from stack:

Algorithm

```
POP(STACK, TOP, ITEM)
Step 1:   [Check whether the stack is empty]
          If TOP = 0
          then print "Stack underflow" and exit.
Step 2:   [Assigns TOP element to ITEM]
          ITEM = STACK[TOP]
Step 3:   [Decrement the TOP value by one]
          TOP = TOP - 1
Step 4:   Exit.
```

Traversing the stack

Traversing means accessing and processing (frequently called ***visiting***) each element of the stack exactly once. The following algorithm traverses a stack STACK.

Algorithm

```
TRAVERSE(STACK, TOP, MAXSTACK)

Step 1:   [Check for stack underflow]
          If TOP = 0
          Then print "Stack underflow" and exit.

Step 2:   [Initialize counter]
          Set K = TOP

Step 3:   Repeat step 4 and 5 while K > 0;

Step 4:   [Visit element]
          Apply PROCESS to STACK[K]

Step 5:   [Decrease counter]
          Set K = K - 1

         [End of Step 3 loop]

Step 6:   Exit.
```

- **Algorithm**

Algorithm is a step-by-step method of solving a problem or making decisions.

Characteristics of Algorithms:

Following are the five important characteristics of an algorithm:

1) Finiteness:

An algorithm must terminate after a finite number of steps and further each step must be executable in finite amount of time.

2) Definiteness (no ambiguity):

Each steps of an algorithm must be precisely defined; the action to be carried out must be rigorously and unambiguously specified for each case.

3) Inputs:

An algorithm has zero or more but only finite, number of inputs.

4) Output:

An algorithm has one or more outputs. The outputs have specific relation to the inputs, where the relation is defined by the algorithm.

5) Effectiveness:

An algorithm should be effective. This means that each of the operation to be performed in an algorithm must be sufficiently basic that it can, in principle, be done exactly and in a finite length of time, by person using pencil and paper.

- **Differences between Linear and Non-Linear data structures:**

Linear data structure	Non-Linear data structure
1. In a linear data structure, data elements are arranged in a linear order where each and every element is attached to its previous and next adjacent.	In a non-linear data structure, data elements are attached in hierarchically manner.
2. In linear data structure, all data elements are present at a single level.	In non-linear data structure, data elements are present at multiple levels.
3. Linear data structures can be traversed completely in a single run.	Non-linear data structures are not easy to traverse and needs multiple runs to be traversed completely.
4. In a linear data structure, memory is not utilized in an efficient way.	In a non-linear data structure, memory is utilized in an efficient way.
5. Its examples are: array, stack, queue, linked list, etc.	Its examples are: trees and graphs.

■ ■ ■ ■ ■

Sant Gadge Baba Amravati University, Amravati
B. Sc. Part ONE (Semester – II) CBCS Examination
Questions Asked in Previous University Exams

- **Summer 2023 (AD-4624)**

2. a) What is stack? Explain algorithm to insert and remove elements from the stack. 7
   b) What are different operations performed on the data structure? Explain. 3

OR

3. a) What is data structure? Explain the types of data structure with suitable example. 7
   b) Explain an algorithm to delete an element from the array. 3

- **Winter 2023 (AE-4606)**

2. A) What is stack? Explain the concept of stack. 3
   B) What is array? Explain the concept of array and its memory representation. 7

OR

3. A) Define & explain Primitive & Non-Primitive data structure with example. 7
   B) Explain the operations performed on Data Structure. 3

■ ■ ■ ■ ■