Data Structures – CST 201
Module ~ 3

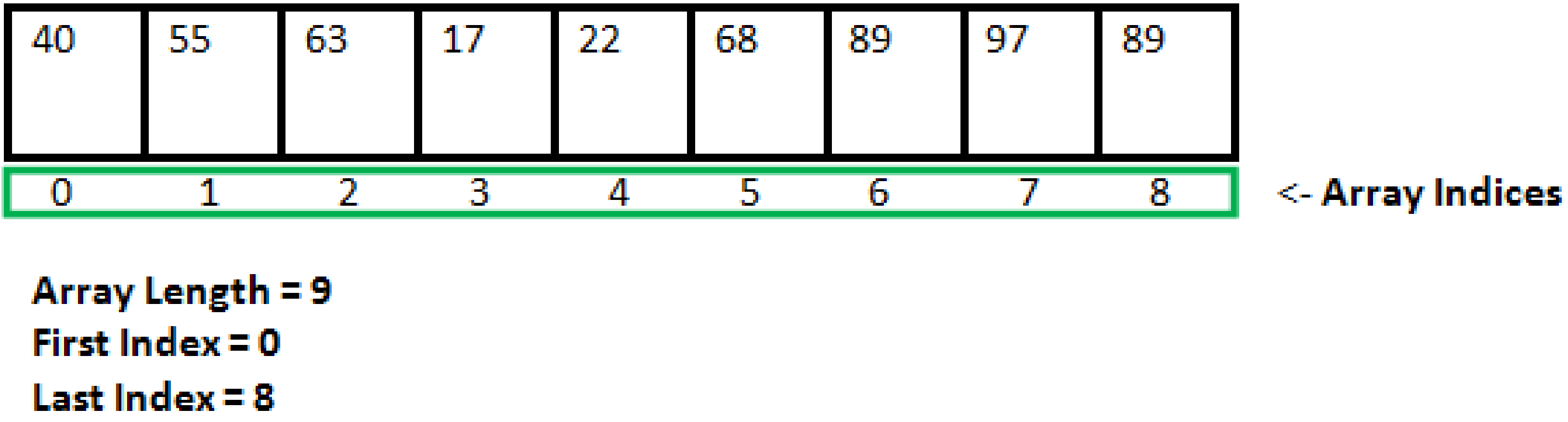
Syllabus

- **Linked List and Memory Management**
  - Self Referential Structures
  - **Dynamic Memory Allocation**
  - Singly Linked List-Operations on Linked List.
  - Doubly Linked List
  - Circular Linked List
  - Stacks using Linked List
  - Queues using Linked List
  - Polynomial representation using Linked List
  - Memory allocation and de-allocation
    - First-fit, Best-fit and Worst-fit allocation schemes

DYNAMIC MEMORY ALLOCATION

INTRODUCTION

- Since C is a structured language, it has some fixed rules for programming.
- One of it includes changing the size of an array.
- An array is collection of items stored at continuous memory locations.



40	55	63	17	22	68	89	97	89	
0	1	2	3	4	5	6	7	8	<- Array Indices

Array Length = 9
First Index = 0
Last Index = 8

INTRODUCTION

- As it can be seen that the length (size) of the array above made is 9. But what if there is a requirement to change this length (size).
- For Example,
  - If there is a situation where only 5 elements are needed to be entered in this array. In this case, the remaining 4 indices are just wasting memory in this array. So there is a requirement to lessen the length (size) of the array from 9 to 5.
  - Take another situation. In this, there is an array of 9 elements with all 9 indices filled. But there is a need to enter 3 more elements in this array. In this case 3 indices more are required. So the length (size) of the array needs to be changed from 9 to 12.
- This procedure is referred to as **Dynamic Memory Allocation in C**.

DYNAMIC MEMORY ALLOCATION

- Therefore, C **Dynamic Memory Allocation** can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.
- C provides some functions to achieve these tasks.
- There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming.
- They are:
  - malloc()
  - calloc()
  - free()
  - realloc()

malloc()

- **“malloc”** or **“memory allocation”** method in C is used to dynamically allocate a **single large block of memory** with the specified size.
- It returns a pointer of type void which can be cast into a pointer of any form.
- It initializes each block with **default garbage value**.

Syntax:
ptr = (data type*) malloc(byte-size)

- For Example:

```
ptr = (int*) malloc(100 * sizeof(int));
```

  - Since the size of int is 2 bytes, this statement will allocate 200 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

calloc()

- **“calloc”** or **“contiguous allocation”** method in C is used to dynamically allocate the **specified number of blocks of memory** of the specified type.
- It initializes each block with **a default value ‘0’**

Syntax:

```
ptr = (data-type*)calloc(n, element-size);
```

- **For Example:**

```
ptr = (float*) calloc(25, sizeof(float));
```

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

free()

- **“free”** method in C is used to dynamically **de-allocate** the memory.
- The memory allocated using functions malloc() and calloc() is not de-allocated on their own.
- Hence the free() method is used, whenever the dynamic memory allocation takes place.
- It helps to reduce wastage of memory by freeing it.

Syntax:

- free(ptr);

realloc()

- **“realloc”** or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory.
- In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**.
- re-allocation of memory maintains the already present value and new blocks will be initialized with default garbage value.
- **Syntax:**

ptr = realloc(ptr, newSize);

- where ptr is reallocated with new size 'newSize'.

Arrays

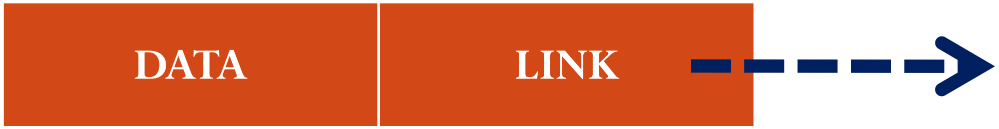
- Arrays are used to store data elements in memory
- **Advantage:**
  - Elements can be accessed fastly
- **Disadvantages:**
  - **Insertion** and **deletion** is relatively expensive.
  - It is a **static data structure**. Array size is fixed. Memory resizing is not possible.
  - Array require **continuous memory locations** to store data.
- So, a new data structure**(Linked List)** is introduced to overcome these disadvantages.

Linked List

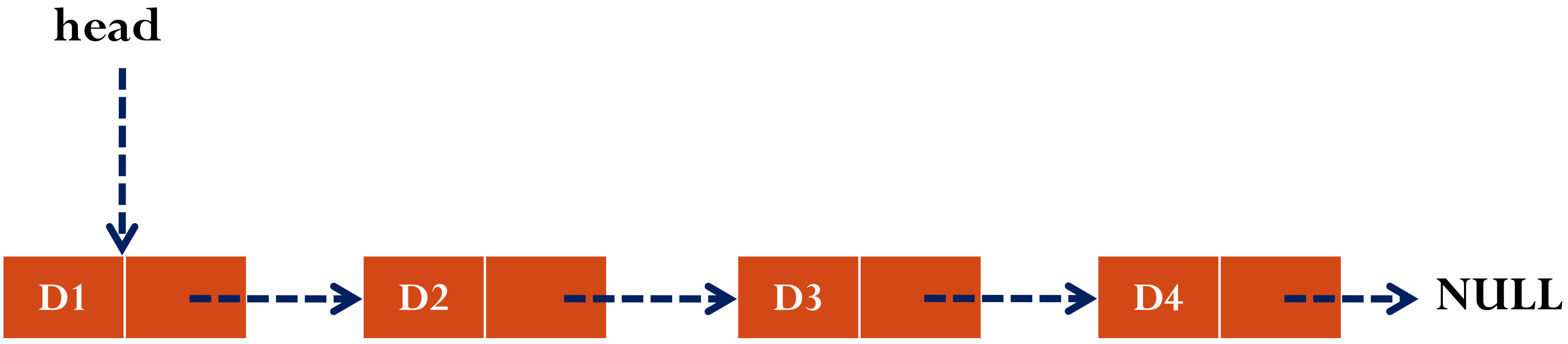
- **Linked list is a dynamic data structure:** Amount of memory required can be varied during its use
- A Linked List is an **Ordered Collection of homogenous elements** where the **linear ordering** is maintained using **links or pointers**
- A linked list can grow or shrink in size as the program runs
- **Insertion** and **deletion** can be performed **fastly**.

Linked List

- Element in a linked list is termed as **node**
- A node consist of two fields
  - **DATA:** To store the actual information
  - **LINK/POINTER:**
    - Used to **point to the next node.**
    - It is actually an **address of subsequent element**
    - In linked list adjacency between the elements are maintained by means of **links/pointers**



Linked List



- A linked list is a **series of connected nodes**
- Each node contains at least
  - Data (any type)
  - Pointer to the next node in the list
- **head**: pointer to the first node
- The last node points to **NULL**

Linked List

- Linked list can be classified into 4 groups
  - Singly linked list
  - Doubly linked list
  - Circular linked list
  - Circular Doubly linked list