Data Structure

Tarkeshwar Barua

Introduction

- A data structure is a specific way of storing and organizing data so that it can be used effectively and efficiently.
- An efficient data structure takes up little memory space and requires as little time as possible to execute the data.
- A data structure is a particular way of organising data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks

Major Operations

- Traversing
  - Accessing each record exactly once so that certain items in the record may be processed.
- Searching
  - Finding the location of the record with a given key value, or finding the locations of all records that satisfy one or more conditions.
- Inserting
  - Adding a new record to the structure.
- Deleting
  - Removing a record from the structure.

Advantages of Data Structure

- Data structures allow information storage on hard disks.
- provides means for management of large dataset such as databases or internet indexing services.
- Are necessary for design of efficient algorithms.
- Allows safe storage of information on a computer.
- Allows the data use and processing on a software system

Where to use?

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

Example of Data Structure

- RDBMS uses Array data structure
- Network data model uses Graph
- Hierarchal data model uses Trees

Types of Data Structure

- Linear Data Structure
  - Static Data Structure(Array)
  - Dynamic Data Structure(Queue, Stack, LinkedList)
  - Elements are arranged in one dimension ,also known as linear dimension.
  - Example: lists, stack, queue, etc.
- Non-Linear Data Structure.
  - Elements are arranged in one-many, many-one and many-many dimensions.
  - Example: tree, graph, table, etc.
  - Tree
  - Graph

Linear Data Structure

- A data structure is called linear if all of its elements are arranged in the sequential order. In linear data structures, the elements are stored in a non-hierarchical way where each item has the successors and predecessors except the first and last element

Non-Linear Data Structure

- The Non-linear data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement.
- The data elements are not arranged in the sequential structure.

Built-in Data Structures in Java

Inbuild Data Structure	Internal Working
ArrayList	Array of Object classes
LinkedList	List and Deque
Stack	Vector class
Queue	LinkedList class
PriorityQueue	Binary Heap
HashSet	HashMap
HashMap	Array of buckets
TreeSet	TreeMap
TreeMap	Red-black tree
HashTable	Slot/Bucket
LinkedHashSet	HashTable and LinkedList

Arrays

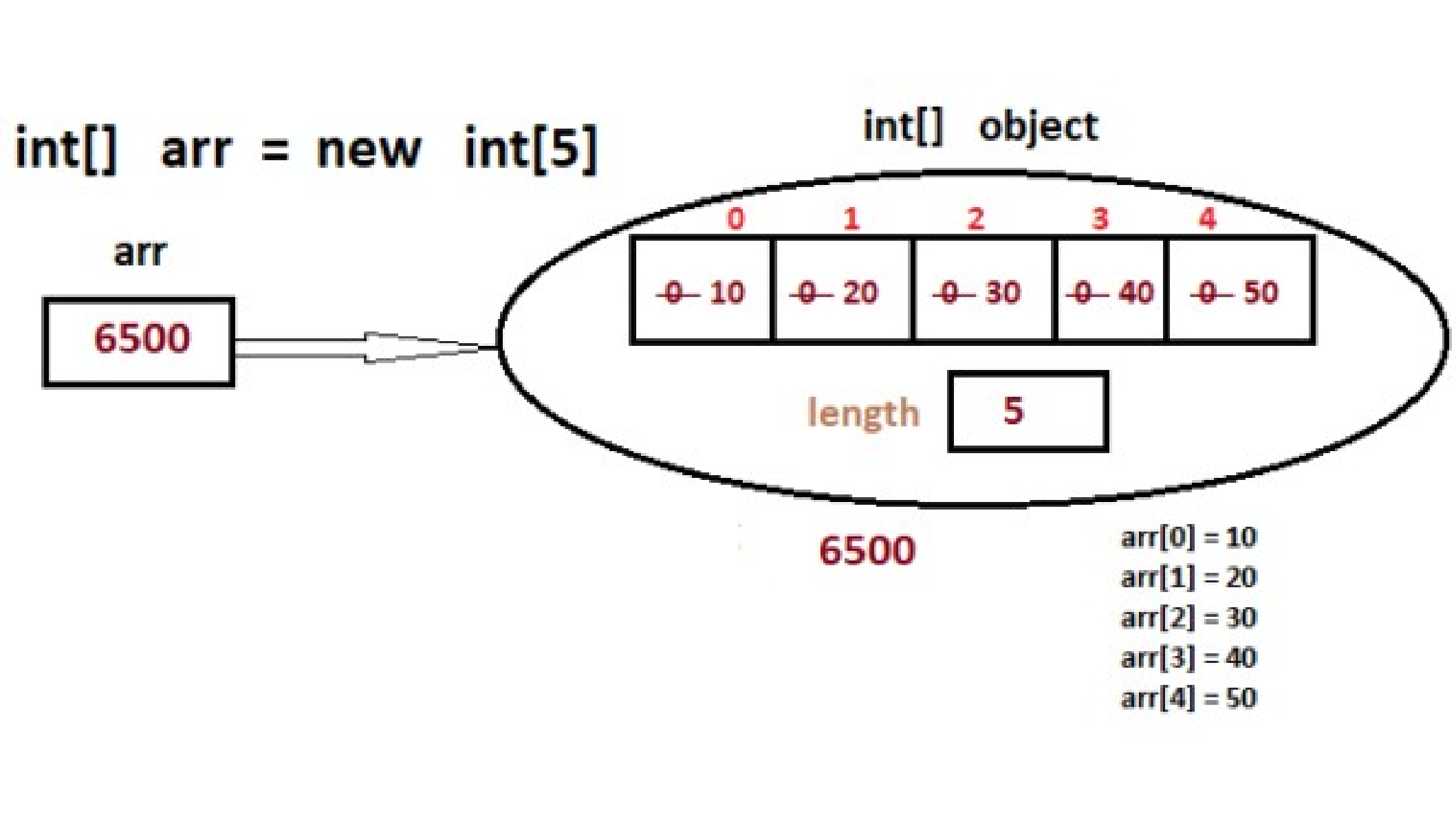
An ordered collection of items is an array

Introduction

- Array items are typically stored in a sequence of computer memory locations
- A convenient way to write them down on paper.
- We can just write the items in order, separated by commas and enclosed by square brackets.

[1, 4, 17, 3, 90, 79, 4, 6, 81]

a = [1, 4, 17, 3, 90, 79, 4, 6, 81]


int[] arr = new int[5]
int[] object
arr
6500
0
1
2
3
4
0 10
0 20
0 30
0 40
0 50
length
5
6500
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50

```java
package arrays_demo;

public class ArrayLength {
    public static void main(String[] args) {
        // declare and initialize an array
        int arr[] = { 10, 20, 30, 40, 50 };

        // display array length
        System.out.print("The length of the given array = ");
        System.out.println(arr.length);
    }
}
```

Sum of Array

- array[] = {10, 20, 30}
- Sum of array elements = 10+20+30 = 60

- Similarly,
- array[] = {50, 60, -20, 55, -90}
- Sum of array elements = 50+60-20+55-90 = 55

Steps to find Sum of Array

- Take one array.
- Declare one sum variable and initialize it with 0.
- Using a loop, Iterate through the elements of the given array.
- Add element to the sum variable and assign result value back to the sum variable. (i.e. sum = sum + arr[i])
- When all elements of the array are added to the sum variable then display the result.

```
// declare array and
// initialize it with values
int array[] = { 10, 20, 30, 40, 50 };
// initialize sum variable with 0
int sum = 0;
// add array elements
for (int i = 0; i < array.length; i++) {
    sum += array[i]; // sum = sum + array[i];
}
// display the result
System.out.println("Sum of array elements= " + sum);
```

Question?

- Please write a program to find average of Array
- Please write a program to Get Array Input in Java
- Please write a program to Return Array from a method
- Please write a program to Different ways to Print Array
- Please write a program to Find the Sum of Array in Java
- Please write a program to Find the average of an Array
- Please write a program to Sum of Two Arrays Elements

```
public class ArrayLengthTwoDimentional {
    public static void main(String[] args) {
      // declare and initialize an array
      int arr[][] = { { 50, 60 }, { 70, 80 }, { 90, 100 } };
     // display array length
      System.out.print("The length of the given array = ");
      System.out.println(arr.length);
      System.out.println("arr[0] length = " + arr[0].length);
      System.out.println("arr[1] length = " + arr[0].length);
      System.out.println("arr[2] length = " + arr[0].length);
    }
}
```

```
// 2d array
        int[][] a = { { 10, 20 }, { 30, 40 }, { 50, 60 } };
        // display size of array
        System.out.println("2D array size = " + a.length);
        System.out.println("First row size = " + a[0].length);
        System.out.println("Second row size = " + a[1].length);
        // display array using length property
        System.out.println("Array elements:");
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                System.out.print(a[i][j] + "\t");
      }
          System.out.println();
    }
   }
}
```

Sum of Two Arrays Elements

- array1[] = {10, 20, 30, 40, 50};
- Array2[] = {9, 18, 27, 36, 45};

- The resultant array will be,
- array3[] = {19, 38, 57, 76, 95};
- And it was calculated as,
- array3[] = {10+9, 20+18, 30+27, 40+36, 50+45};

```
// create Scanner class object
Scanner scan = new Scanner(System.in);
// take number of elements in both array
System.out.print("Enter number of elements in first array: ");
int array1size = scan.nextInt();
System.out.print("Enter number of elements in second array: ");
int array2size = scan.nextInt();
// both array must have same number of elements
if (array1size != array2size) {
System.out.println("Both array must have " + "same number of elements");
return;
}
// declare three array with given size
int array1[] = new int[array1size];
int array2[] = new int[array1size];
int array3[] = new int[array1size];
```

```
// take input for array1 elements
System.out.println("Enter first array elements: ");
for (int i = 0; i < array1.length; i++) {
array1[i] = scan.nextInt();
}
// take input for array2 elements
System.out.println("Enter second array elements: ");
for (int i = 0; i < array2.length; i++) {
array2[i] = scan.nextInt();
}
// loop to iterate through the array
for (int i = 0; i < array3.length; i++) {
// add array elements
array3[i] = array1[i] + array2[i];
}
// display the third array
System.out.println("Resultant Array: " + Arrays.toString(array3));
```

```
public class ThreeDimentionalArray {

   public static void main(String[] args) {

      // declare and initialize an array

      int[][][] arr = { { { 1, 2 }, { 3, 4 }, { 5, 6 } }, { { 7, 8 },
{ 9, 1 }, { 2, 3 } } };

      // display array length

      System.out.print("The length of the given array = ");

      System.out.println(arr.length);

      System.out.println("arr[0] length = " + arr[0].length);

      System.out.println("arr[0][0] length = " + arr[0][0].length);

      System.out.println("arr[0][1] length = " + arr[0][1].length);

      System.out.println("arr[0][2] length = " + arr[0][2].length);

   }

}
```

```
import java.util.Scanner;

public class InputArrays {
    public static void main(String[] args) {

// Scanner class object to read input
        Scanner scan = new Scanner(System.in);
        // declaring and creating array objects
        int[] arr = new int[5];
        // displaying default values
        System.out.println("Default values of array:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + "\t");
        }
        // initializing array
        System.out.println("\n\n***Initializing Array***");
        System.out.println("Enter " + arr.length + " integer values:");
        for (int i = 0; i < arr.length; i++) {
            // read input
            arr[i] = scan.nextInt();
        }
        System.out.println("***Initialization completed***\n");
        //displaying initialized values
        System.out.println("Array elements are:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + "\t");
        }
    }
}
```

Lists

ArrayList

- The backing data structure of ArrayList is an array of Object classes.
- Internally an ArrayList uses an Object[] Array which is an array of objects.
- All operation like iteration, deleting, adding, and updating the elements happens in this Object[] array.

LinkedList

- Java’s LinkedList class is a versatile and commonly used data structure that implements the List and Deque interfaces while extending the AbstractSequentialList

How to find n’th element from the last, if size of the LinkedList is unknown? (asked in Morgan Stanley)

Nth-to-Last Element in a Singly Linked List

Types of Linked List

- Single Linked Lists(SLL)
- Circular Linked List(CLL)
- Doubly Linked List(DLL)
- Differences between Linked Lists vs. Arrays

Applications of Linked List

Stepping stone to implementing graphs.

Useful for playing video and sound files in “looping” mode.

Used to implement associative arrays, and are in this context called association lists.

Used as a building block for many other data structures, such as stacks, queues, self-balancing binary search trees . and their variations.

To recognize a palindrome, a queue can be used in conjunction with a stack.
A stack can be used to reverse the order of occurrences.
A queue can be used to preserve the order of occurrences.

Singly-Linked List (SLL)

SLL is a linear DS and collection of nodes, connected by the links in any order with one direction is called Singly Linked Lists.
Single linked list is a sequence of elements in which every element has link to its next element in the sequence, the simplest kind of linked list is a singly-linked list, which has one link per node.

It is simple sequence of dynamically allocated Nodes.

Each Node has its successor and predecessor.

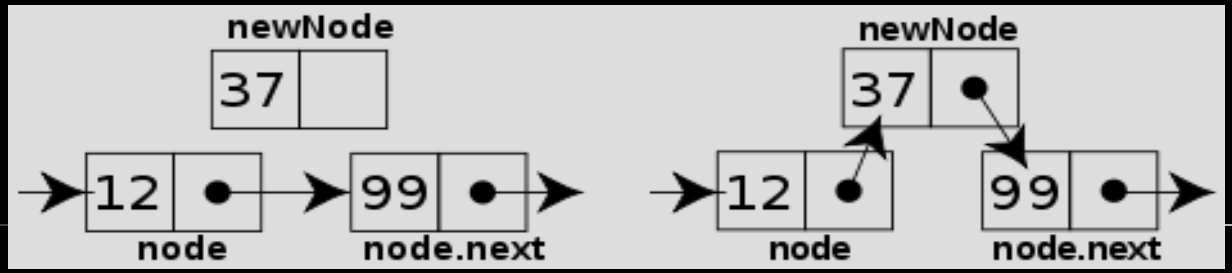
In a single linked list, the address of the first node is always stored in a reference node known as "front" or "head"), always next part i.e. reference part of the last node must be NULL.

Singly-Linked List (SLL)

A **linked list** is a way to store a collection of elements, like an array these can be character or integers, each element in a linked list is stored in the form of a **node**.

Each node contains a value and a link to its successor (the last node has no successor).

The header points to the first node in the list (or contains the null link if the list is empty)

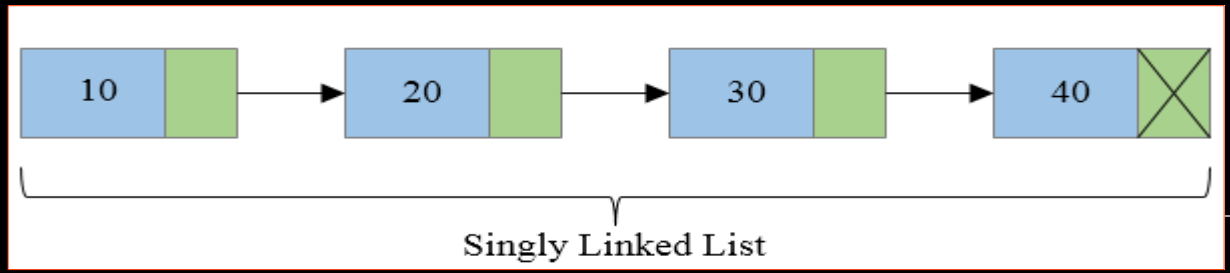


Singly-Linked List (SLL)

Begins with a object to the first node.
Terminates with a null.
Only traversed in one direction.

In this type of Linked List two successive nodes are linked together in linear fashion .
Each Node contain address of the next node to be followed.

In Singly Linked List only Linear or Forward Sequential movement is possible.
Elements are accessed sequentially , no direct access is allowed.



Singly-Linked List

Last Node have successor reference as “NULL“.

In the above Linked List We have 3 types of nodes.

First Node

Last Node

Intermediate Nodes

In Singly Linked List access is given only in one direction thus Accessing Singly Linked is Unidirectional.

We can have multiple data fields inside Node but we have only single “Link” for next node.

In this type of Linked List two **successive nodes** are linked together in **linear fashion** .

Each Node contain **address of the next node** to be followed.

In Singly Linked List only Linear or Forward **Sequential movement is possible**.

Elements are accessed sequentially , **no direct access** is allowed.

Singly Linked List: Insertion

- Problem: Insert a new element at a given point in a linked-list.
- Four cases to consider:
- insertion in an empty linked-list;
- insertion before the first node of a non-empty linked-list;
- insertion after the last node of a non-empty linked-list;
- insertion between nodes of a non-empty linked-list.
- The insertion algorithm needs links to the new node's successor and predecessor.

Singly Linked List: Deletion

- Problem: Delete a given node from a linked-list.
-
- Four cases to consider:
- deletion of a singleton node;
- deletion of the first (but not last) node;
- deletion of the last (but not first) node;
- deletion of an intermediate node.
    -
- The deletion algorithm needs links to the deleted node's successor and predecessor.

Singly Linked List: Deletion

Advantages: Store items "sequentially" without restrictions on location.

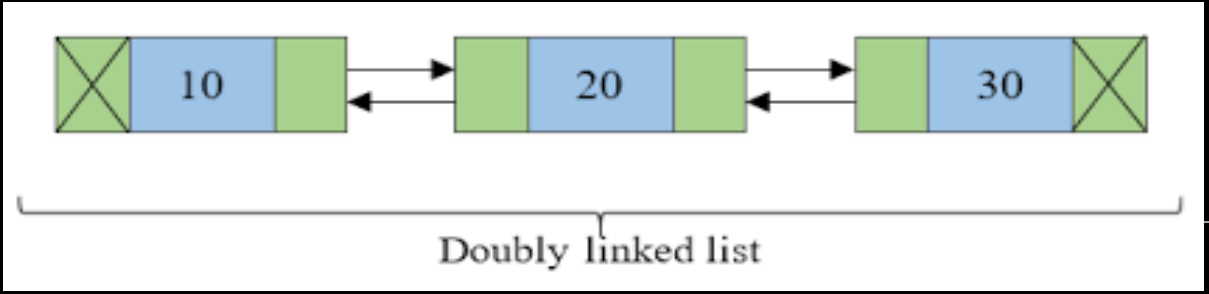
- Access any item as long as external link to first item maintained.
- Insert new item without shifting.
- Delete existing item without shifting.
- Size can expand/contract throughout use.

Disadvantages

- Consumes more memory.
- Overhead of links
- No longer have direct access to each element of the list.
- O(1) access becomes O(n) access since we must go through first element, and then second, and then third, etc.
- If dynamic, must provide destructor, copy constructor(memory leak).

Doubly-Linked List (DLL)

- DLL is a linear data structures and collection of nodes, connected by links in both directions is called DLL.
-
- Doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.

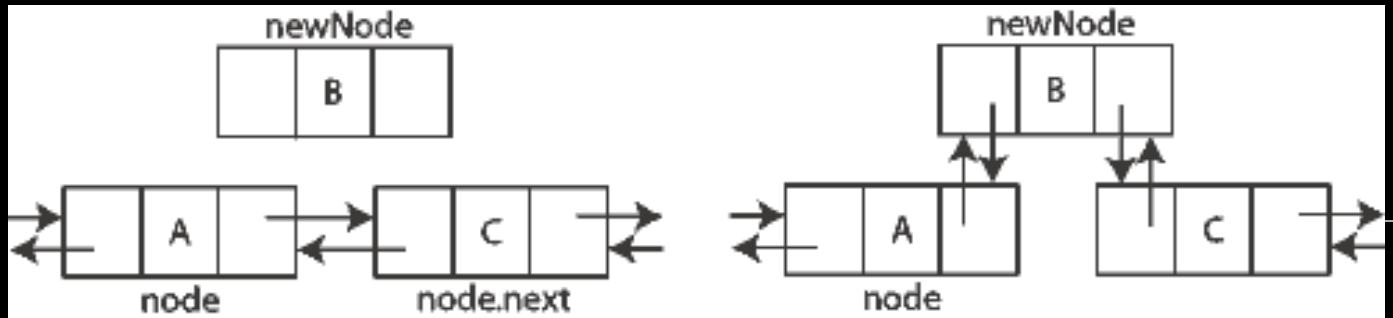


Doubly-Linked List (DLL)

- Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.
-
- **Link** – Each link of a linked list can store a data called an element.
- **Next** – Each link of a linked list contains a link to the next link called Next.
-
- **Prev** – Each link of a linked list contains a link to the previous link called Prev.
  A Linked List contains the connection link to the first link called First and to the last link called Last.

Doubly-Linked List (DLL)

- As per the above illustration, following are the important points to be considered.
- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and two link fields called next and prev.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.



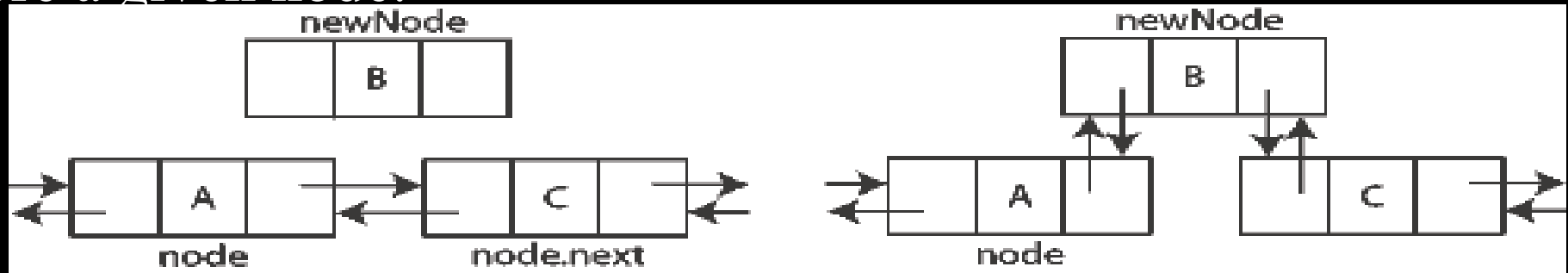
Doubly-Linked List (DLL)

- In a doubly linked list, each node contains three fields,
- Data: which holds the elements of the list.
- Next: which stores a Object (link) to the next node in the list.
- Previous: which stores a Object (link) to the previous node in the list.

- If there is no predecessor (pred) the link is null.
- If there is no successor (succ) the link is null.
- If the list is empty both links in the header are null.

Doubly-Linked List (DLL): Insertion

- Must update references in both predecessor and successor nodes.
- DLL insertion algorithm: To insert element at a given point in the DLL headed by (first, last):
- A node can be added in four ways
  1) At the front of the DLL
  2) After a given node.
  3) At the end of the DLL
  4) Before a given node.



Doubly-Linked List (DLL): Deletion

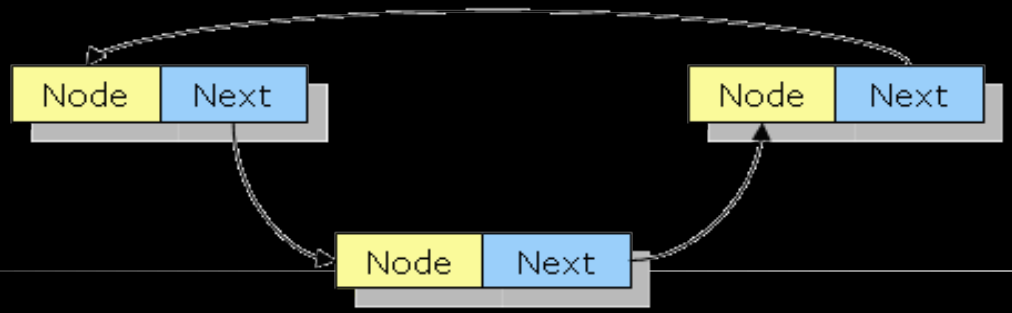
- To delete node del from the DLL headed by (first, last):
- Let the node to be deleted is del.
- If node to be deleted is head node, then change the head pointer to next current head.
- Set next of previous to del, if previous to del exists.
- Set prev of next to del, if next to del exists.
  - **ADVANTAGES**
- Doubly linked lists are useful for playing video and sound files with “rewind” and “instant replay”.
- They are also useful for other linked data which require “rewind” and “fast forward” of the data.

Circular Linked Lists (CLL)

- Circular Linked List is a Linear Data Structures, is a collection of nodes and connected by the links, the last node points to the first node is called CLL.
- In a **circularly linked list**, all nodes are **linked** in a continuous circle, without using null. For **lists** with a front and a back (such as a queue), one stores a reference to the last node in the **list.** The next node after the last node is the first node.
-
- In Circular Linked List Address field of Last node contain address of “First Node“.
- In short First Node and Last Nodes are adjacent.
- Linked List is made circular by linking first and last node, so it looks like circular chain.
- Two way access is possible only if we are using “Doubly Circular Linked List”.
- Sequential movement is possible and No direct access is allowed.

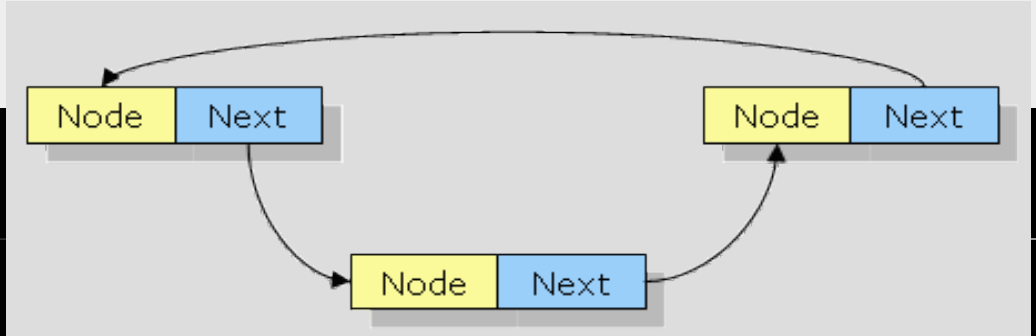
Circular Linked Lists (CLL)

- Properties
- Can reach entire list from any node.
- Need special test for end of list.
- Represent
- Buffers.
- Naturally circular data.



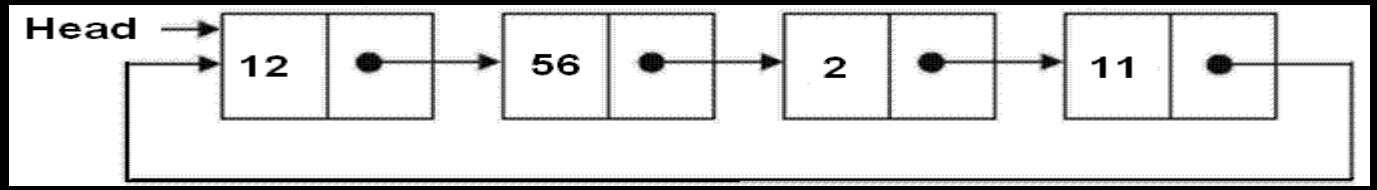
Circular Linked Lists (CLL)

- A good example of an application where **circular linked list** should be **used** is a timesharing problem solved by the operating system.
- A **circularly linked list** node looks exactly the same as a linear singly **linked list.**
-
- Circular Linked List is Divided into 2 Categories .
- Singly Circular Linked List
- Doubly Circular Linked List
  -
  - **1. Singly Circular Linked List:** A singly **linked circular list** is a **linked list** where the last node in the list points to the first node in the **list**.



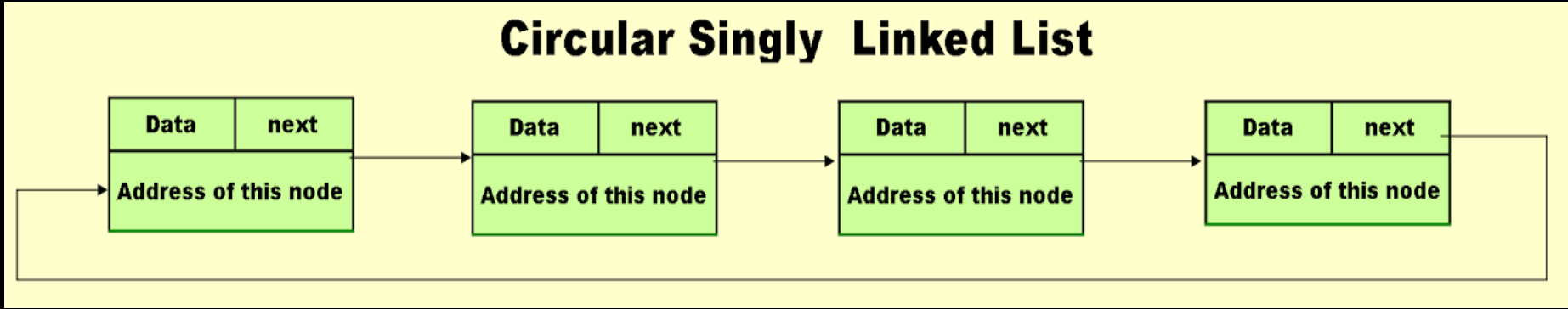
Circular Linked Lists (CLL)

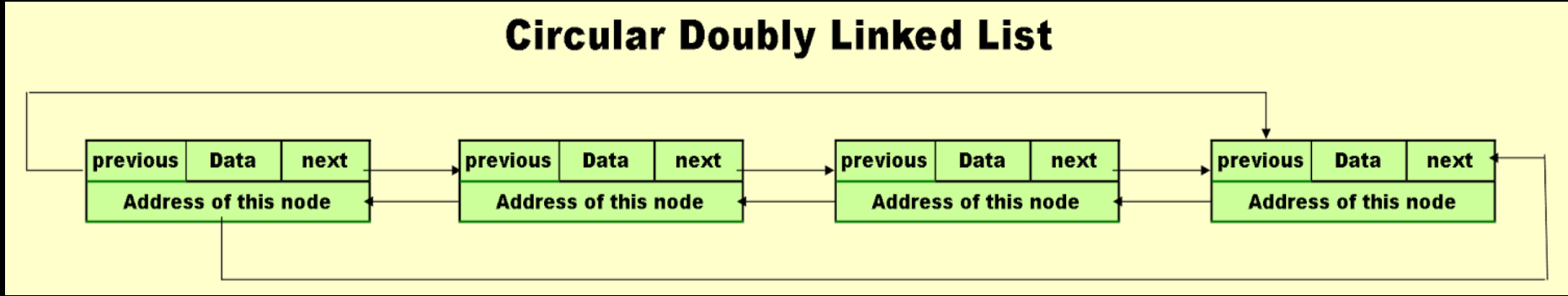
- A **circular list** does not contain NULL pointers.
- Object in the last node points
- Back to the first node.
  - **2. Doubly Circular Linked List:** A Doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.
- Forward Object of the last node points to the first node and backward pointer of the first node points to the last node.



Doubly Linked Circular list

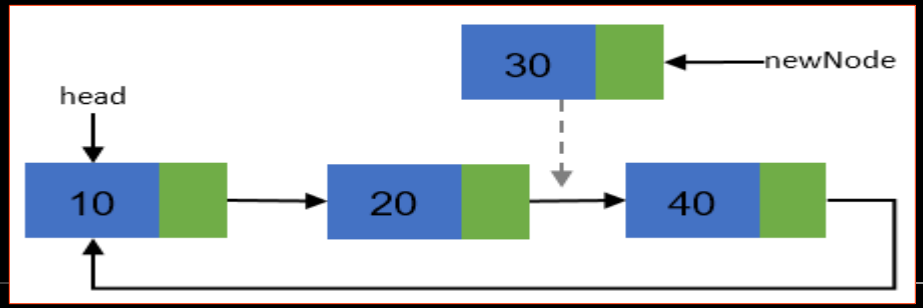
Singly and Doubly Circular Linked List



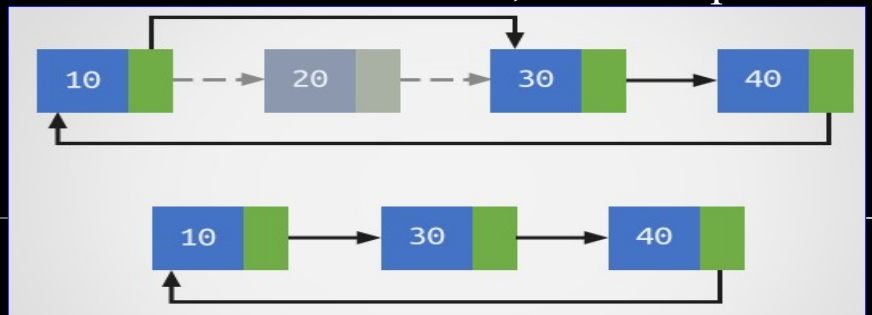


Singly and Doubly Circular Linked List

- **1) Insertion:** A node can be added in three ways:
- Insertion in an empty list
- Insertion at the beginning of the list
- Insertion at the end of the list
- Insertion in between the nodes



- **2) Deletion: Algorithm**
- **Case 1**: List is empty. If the list is empty we will simply return.
-
- **Case 2**:List is not empty
- If the list is not empty then we define two pointers **curr** and **prev** and initialize the pointer **curr** with the **head** node.
- Traverse the list using **curr** to find the node to be deleted and before moving curr to next node, everytime set prev = curr.
- If the node is found, check if it is the only node in the list. If yes, set head = NULL and free(curr).
- If the list has more than one node, check if it is the first node of the list. Condition to check this(curr == head). If yes, then move prev until it reaches the last node. After prev reaches the last node, set head = head -> next and prev -> next = head. Delete curr.
- If curr is not first node, we check if it is the last node in the list. Condition to check this is (curr -> next == head).
- If curr is the last node. Set prev -> next = head and delete the node curr by free(curr).
- If the node to be deleted is neither the first node nor the last node, then set prev -> next = temp -> next and delete curr.



Singly and Doubly Circular Linked List

- **Advantages**
- Each node is accessible from any node.
- Address of the first node is not needed.
- Certain operation, such as concatenation and splitting of string, is more efficient with circular linked list.
- Circular linked lists are useful for playing video and sound files in “looping” mode.
- They are also a stepping stone to implementing graphs.
- Circular linked lists are usually sorted.
-
- **Disadvantage**
- Danger of an infinite loop !

Linked List V/s Arrays

- Elements are stored in linear order, accessible with links.
- Different types of data.
- Do not have a fixed size.
- Cannot access the previous element directly.
- Size and shape can be increase.
- Retrieve the elements based on reference.
- No memory wastage.
- Insertion and deletion will be easy.
- It is a dynamic DS.

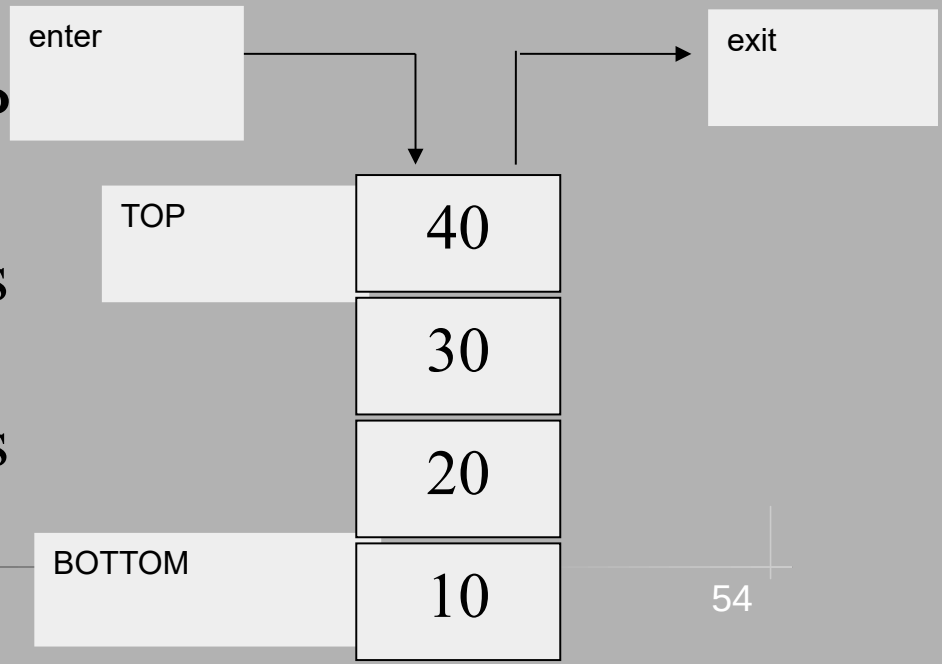
- Elements are stored in linear order, accessible with an index.
- Same types of data.
- Have a fixed size.
- Can access the previous element easily.
- Size and shape can’t be increase.
- Fast retrieval based on index.
- Memory wastage.
- Insertion and deletion will be tough.
- It is a Static DS.

Stack

- The Stack class in Java extends the Vector class, which is a dynamic array-like data structure.
- Stack operations, Stack Linked List, Implementation, Stack applications
- Common stack operations are supported, such as push, pop, peek, and isEmpty.

Stack

- Stack is a Linear Data Structure, in which insertions or deletions will be done at only one end called Top of the Stack.
- Stack follows the principle of LIFO, FILO(First in, Last out), Stack is a LIFO (last in, first out) data structure.
- Insertions into stack is called as PUSH Operation
- Deletions from stack is called as POP Operations
- Whenever the Stack is Full that case is referred to as STACK OVERFLOW.
- If the Stack becomes Empty then it is referred to as STACK UNDERFLOW.



Stack

- Examples:
- Stack of plates in a cafeteria(ldle pates in Container)
- Discard pile in a game of rummy.
- Books arranged in library.
- In computer world function calls
- Recursion calls and returns.
- Evaluation of Arithmetic expressions.
- CD Container.

Stack

- Stack: A list in which entries are removed and inserted only at the head.
- LIFO: Last-in-first-out.
- Top: The head of list (stack).
- Bottom or base: The tail of list (stack).
- Pop: To remove the entry at the top.
- Push: To insert an entry at the top.
- List: A collection of data whose entries are arranged sequentially.
- Head: The beginning of the Stack.
- Tail: The end of the Queue.

Stack Operations

1. Push()	Add an item to the top of the stack
2. Pop()	Remove the item most recently added
3. Display()	Display the elements from the stack
3. isEmpty()	Determine whether the stack is empty
4. isFull()	Determine whether the stack is full
5. Peek()	Retrieve the item most recently added
6.Searching()	Searching an element from the stack
7.Sorting()	Sorting elements from the stack

Applications of Stack

- Parsing.
- Recursive Function.
- Calling Function.
- Towers of Hanoi.
- Elevator Algorithm.
- Storing Local Variables
- Web Browsing (Back Operations).
- UNDO Operations in web browsing.
- Calculators, Word Processor(editor) etc.
- Backtracking (game playing, finding paths, exhaustive searching).
- Memory management, run-time environment for nested language features.
- Expression Evaluation
- Expression Conversion
- Infix to Postfix
- Postfix evaluation

Queue

- A queue is a First-In-First-Out (FIFO) data structure used to manage elements in a way that the first element added is the first one to be removed.
- Queue operations, Queue Linked List design, Queue applications.
- The Queue interface can be implemented using LinkedList or ArrayDeque.

PriorityQueue

- A priority queue stores elements with associated priorities, and elements with higher priorities are dequeued first.
- The ordering can be based on natural ordering or defined using a comparator.
- It’s typically implemented using a binary heap or another data structure that maintains the priority order.

Hashing

Sets

HashSet

- HashSet internally uses HashMap to store it’s elements.
- Whenever you create a HashSet object, one HashMap object associated with it is also created.

HashMap

- HashMap in Java is basically an array of buckets. where each bucket uses linked list to hold elements.
- A bucket is a linked list of nodes where each node is an object of class Node<K,V>.

Binary Tree

Binary Search Tree

TreeSet

- The TreeSet internally uses the TreeMap to store the elements.
- Whenever we create a TreeSet, The JVM internally creates a TreeMap and perform all the operations on TreeMap.

TreeMap

- TreeMap is a red-black tree-based implementation of the Map interface, NavigableMap, and AbstractMap classes.

HashTable

- Hash table intrinsically contains a slot/bucket in which the storage of key and value pair.
- It uses the key’s hash code to discover which bucket the key/value of a set should map.

LinkedHashSet

- LinkedHashMap in Java is an implementation that combines HashTable and LinkedList implementation.
- It implements the Map interface.

Dictionaries

Graphs

Graph Algorithms

Sorting Algorithms

Searching Algorithms

Advanced Algorithms