2.1

Arrays Lists and Stack

Objectives

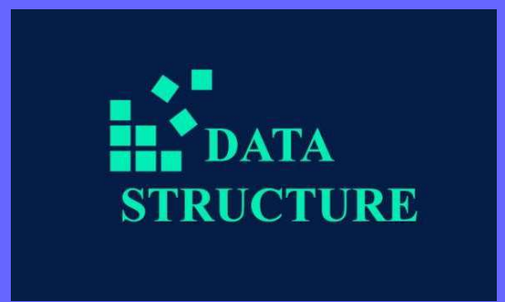
After completing this lesson, you should be able to:

- Recursion
- Singly Linked List, Circular Linked List, Doubly LL
- Factorial

Course Roadmap

Data Structures

- Lesson 1: Introduction to Data Structures
- **Lesson 2: Lists and Stacks** — **You are here!**
- Lesson 3: Searching
- Lesson 4: Trees and Queues
- Lesson 5: Sorting



Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
- It makes the code compact but complex to understand.

Syntax:

```
returntype methodname(){
//code to be executed
methodname();//calling same method
}
```

Types of Recursion:

1. A function is called direct recursive if it calls the same function.
2. A function **fun** is called indirect recursive if it calls another function say **fun_new** and **fun_new** calls **fun** directly or indirectly.

Direct recursion:

```
void directRecFun()
{
    // Some code....

    directRecFun();

    // Some code...
}
```

Indirect recursion:

```
void indirectRecFun1()
{
    // Some code...

    indirectRecFun2();

    // Some code...
}

void indirectRecFun2()
{
    // Some code...

    indirectRecFun1();

    // Some code...
```

Java Recursion Example : Infinite times

```
public class Recursion1 {
        static void p(){
                System.out.println("hello");
                p();
        }

        public static void main(String[] args) {
                p();
        }
}
```

- Leads into stack Overflow Error

Java Recursion Example : Finite times

```
public class Recursion2 {

    static int count = 0;

    static void p() {
        count++;
        if (count <= 5) {
            System.out.println("hello " + count);
            p();
        }
    }

    public static void main(String[] args) {
        p();
    }
}
```

Factorial

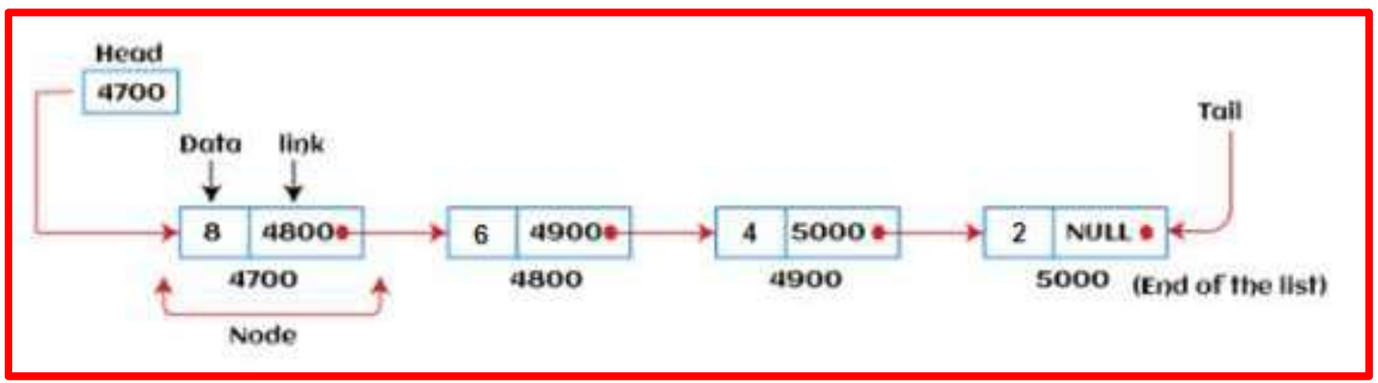
```
public class FactorialDemo {

    static int factorial(int n) {
        if (n == 1) {
            return 1;
        } else {
            return (n * factorial(n - 1));
        }
    }

    public static void main(String[] args) {
        System.out.println("Factorial of 5 is: " + factorial(5));
    }
}
```

Fibonacci Series

```
public class FibonacciDemo {

    static int n1 = 0, n2 = 1, n3 = 0;

    static void printFibo(int count) {
        if (count > 0) {
            n3 = n1 + n2;
            n1 = n2;
            n2 = n3;
            System.out.print(" " + n3);
            printFibo(count - 1);
        }
    }

    public static void main(String[] args) {
        int count = 15;
        System.out.print(n1 + " " + n2);   //printing 0 and 1
        printFibo(count - 2);    //n-2 because 2 numbers are already printed
    }
}
```

Linked List Types

- Linked list is a linear data structure that includes a series of connected nodes. Linked list can be defined as the nodes that are randomly stored in the memory.
- A node in the linked list contains two parts, i.e., first is the data part and second is the address part.
- The last node of the list contains a pointer to the null. After array, linked list is the second most used data structure. In a linked list, every link contains a connection to another link.



Why use linked list over array?

Limitations of arrays

- The size of the array must be known in advance before using it in the program.
- Increasing the size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
- All the elements in the array need to be contiguously stored in the memory. Inserting an element in the array needs shifting of all its predecessors.

Cont…

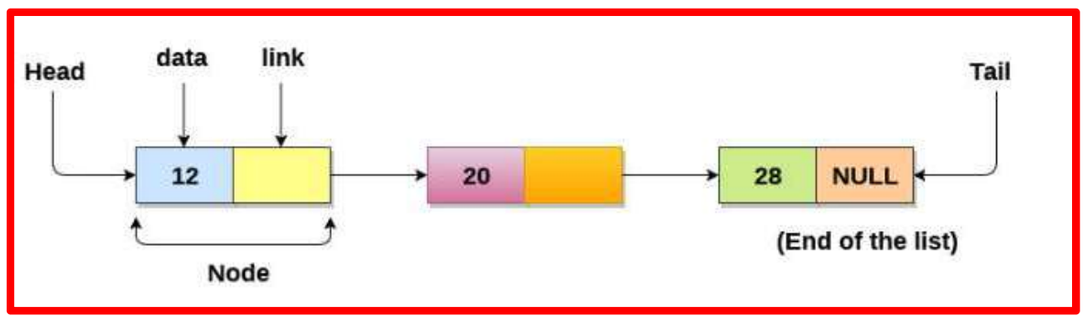
- It allocates the memory dynamically. All the nodes of the linked list are non-contiguously stored in the memory and linked together with the help of pointers.
- In linked list, size is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

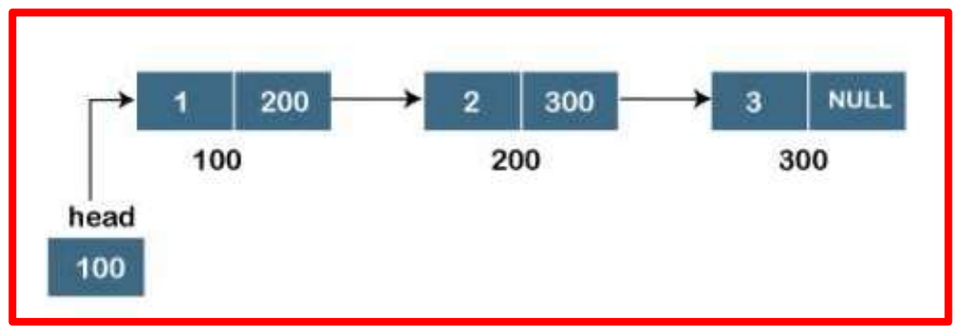
Types of Linked list

1. Singly Linked list
2. Doubly Linked list
3. Circular Linked list

Singly Linked list

- It is the commonly used linked list in programs. If we are talking about the linked list, it means it is a singly linked list. The singly linked list is a data structure that contains two parts, i.e., one is the data part, and the other one is the address part, which contains the address of the next or the successor node.



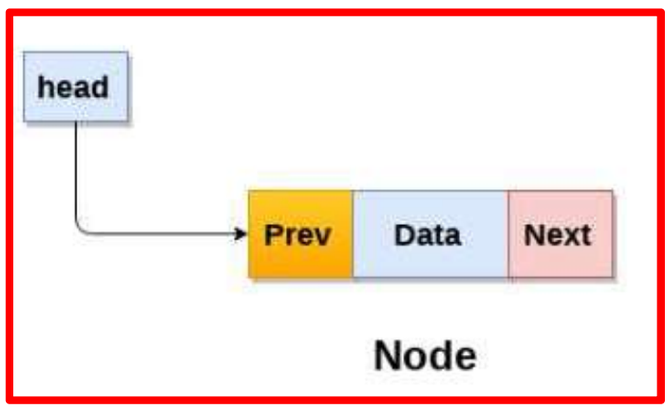


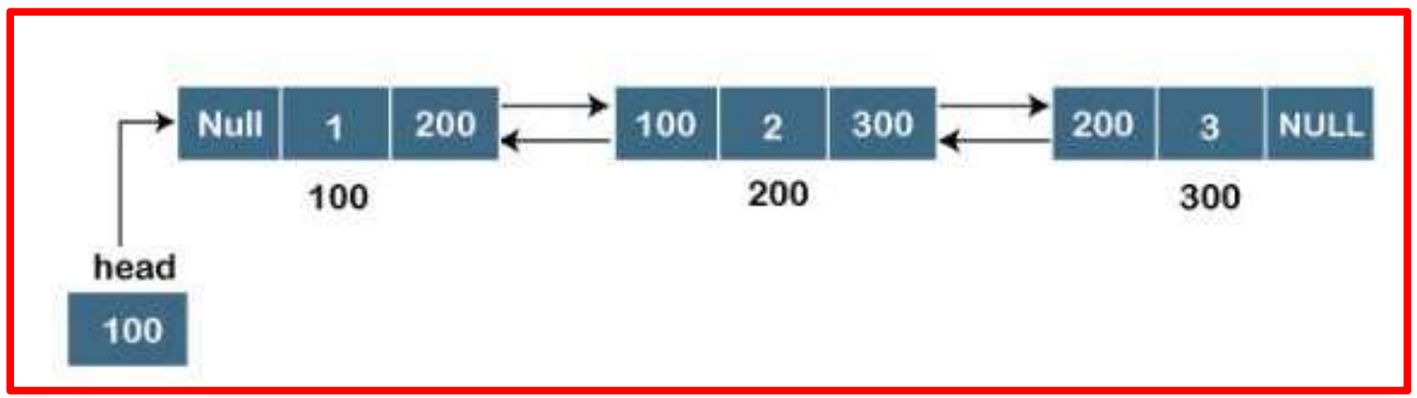
Algorithm

- Create a class Node which has two attributes: data and next. Next is a pointer to the next node.
- Create another class which has two attributes: head and tail.
- addNode() will add a new node to the list:
  - Create a new node.
  - It first checks, whether the head is equal to null which means the list is empty.
  - If the list is empty, both head and tail will point to the newly added node.
  - If the list is not empty, the new node will be added to end of the list such that tail's next will point to the newly added node. This new node will become the new tail of the list.
- a. display() will display the nodes present in the list:
- Define a node current which initially points to the head of the list.
- Traverse through the list till current points to null.
- Display each node by making current to point to node next to it in each iteration.

Doubly linked list

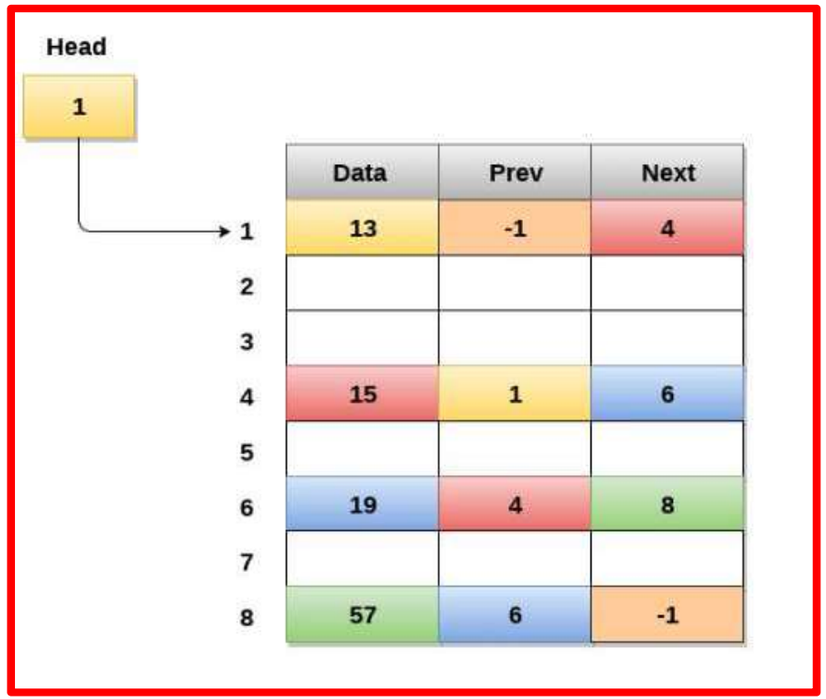
- As the name suggests, the doubly linked list contains two pointers. We can define the doubly linked list as a linear data structure with three parts: the data part and the other two address part.
- In other words, a doubly linked list is a list that has three parts in a single node, includes one data part, a pointer to its previous node, and a pointer to the next node.





Memory Representation of a Doubly linked list

- Generally, doubly linked list consumes more space for every node and therefore, causes more expansive basic operations such as insertion and deletion.
- However, we can easily manipulate the elements of the list since the list maintains pointers in both the directions (forward and backward).

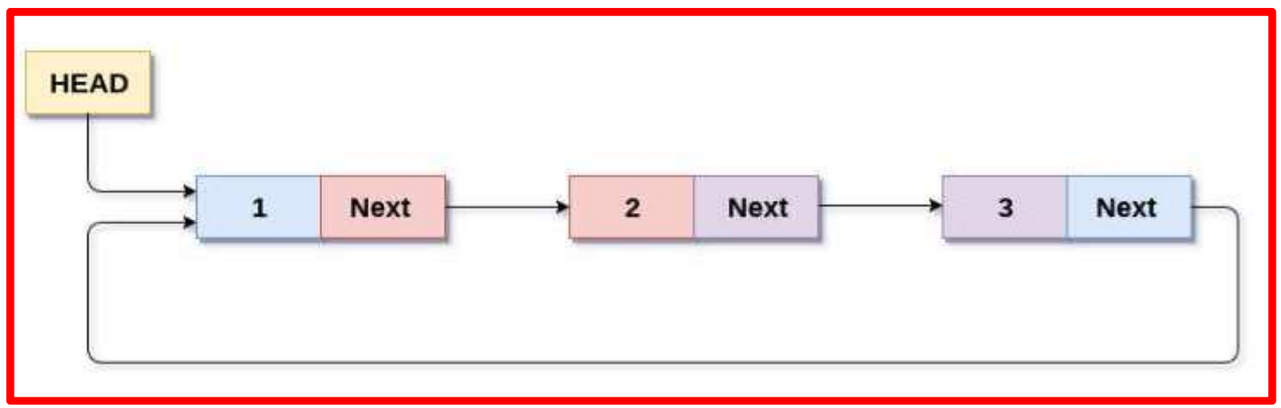


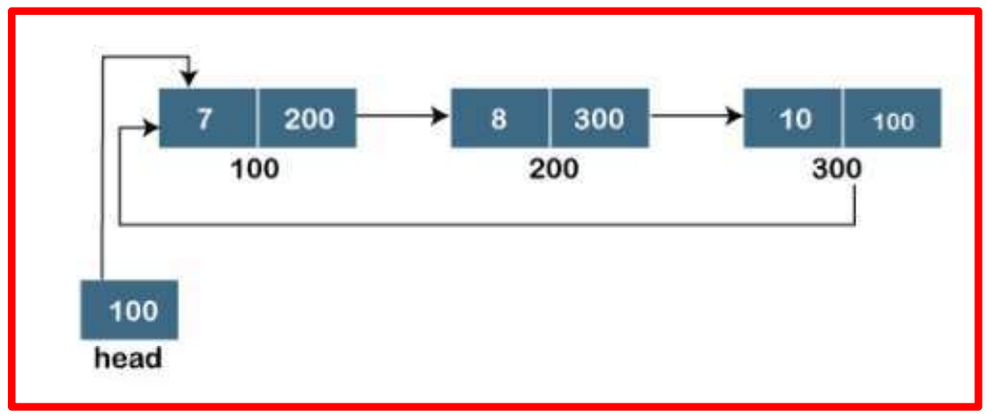
Algorithm

- Define a Node class which represents a node in the list. It will have three properties: data, previous which will point to the previous node and next which will point to the next node.
- Define another class for creating a doubly linked list, and it has two nodes: head and tail. Initially, head and tail will point to null.
- addNode() will add node to the list:
  - It first checks whether the head is null, then it will insert the node as the head.
  - Both head and tail will point to a newly added node.
  - Head's previous pointer will point to null and tail's next pointer will point to null.
  - If the head is not null, the new node will be inserted at the end of the list such that new node's previous pointer will point to tail.
  - The new node will become the new tail. Tail's next pointer will point to null.
- a. display() will show all the nodes present in the list.
- Define a new node 'current' that will point to the head.
- Print current.data till current points to null.
- Current will point to the next node in the list in each iteration.

Circular linked list

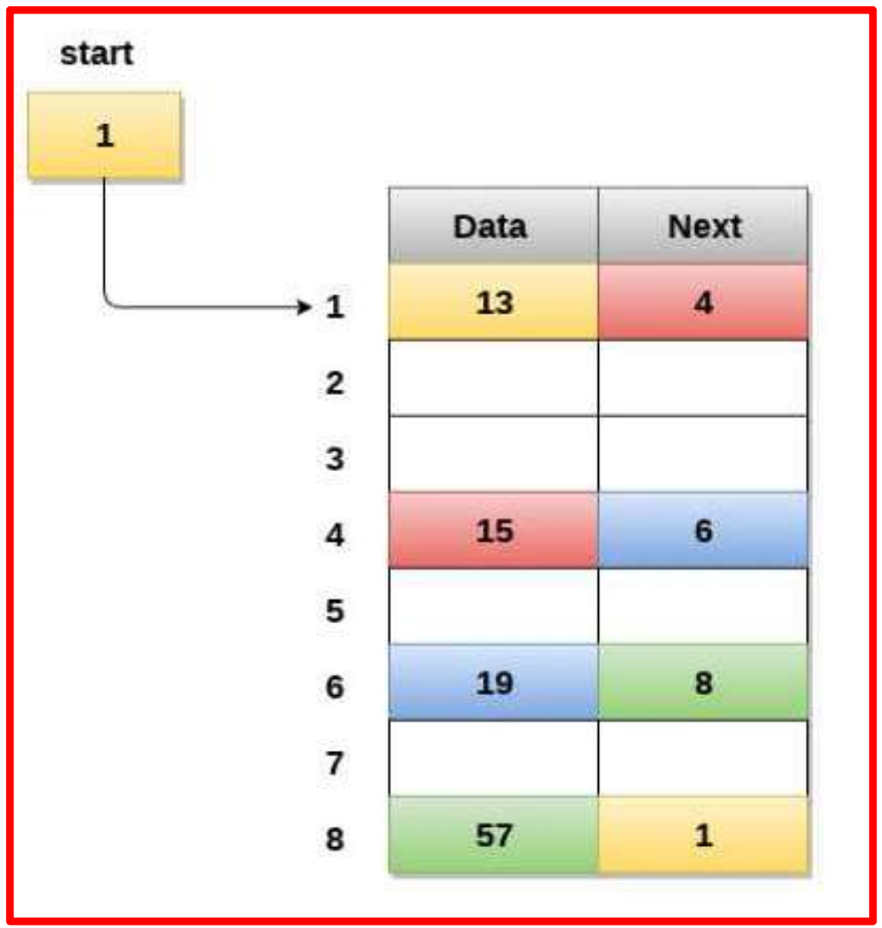
- A circular linked list is a variation of a singly linked list. The only difference between the ***singly linked list*** and a ***circular linked*** list is that the last node does not point to any node in a singly linked list, so its link part contains a NULL value.
- On the other hand, the circular linked list is a list in which the last node connects to the first node, so the link part of the last node holds the first node's address. The circular linked list has no starting and ending node. We can traverse in any direction, i.e., either backward or forward.





Memory Representation of circular linked list:

- Circular linked list are mostly used in task maintenance in operating systems.



LinkedList Class in java.util

```
public class LinkedList1 {

   public static void main(String args[]) {

      LinkedList<String> al = new LinkedList<String>();
      al.add("Rahul");
      al.add("Steven");
      al.add("Imran");
      al.add("Suresh");

      Iterator<String> itr = al.iterator();
      while (itr.hasNext()) {
         System.out.println(itr.next());
      }
   }
}
```

Summary

In this lesson, you should have learned that:

- Recursion
- Singly Linked List, Circular Linked List, Doubly LL
- Factorial