# Lab 2: Linked List from scratch
# 2720 Data Structures

Kiril Kuzmin, Akshay Juyal, Md Abdullah Al Rahat Kutubi

January 19, 2022

Submit on iCollege **by 2pm, January 26, 2022**

PROBLEM 1. [70 POINTS[1]] Implement a linked list of integers as a class LinkedList. Build the following methods:

- ✓ print that prints the content of the linked list;
- ✓ addFirst that adds a new node to the beginning (the head) of the linked list;
- ✓ addLast that adds a new node to the end (the tail) of the linked list;
- ✓ indexOf that finds a specific node by its value, and returns node's index (node's position from the left in the linked list); if the value is not present in the linked list, it returns −1;
- ✓ deleteFirst that deletes the first node in the linked list;
- ✓ deleteLast that deletes the last node in the linked list.

Test your class creating a list in the main and

1) adding one by one nodes 2, 4, 8 to the tail;
2) adding nodes -2, -8 to the head;
3) adding a node 9 to the tail;
4) printing the list;
5) printing indexOf(4);
6) printing contains(9);
7) deleting one by one all the nodes in the list – either from the tail or from the head – and printing the result after each deletion.

In Java, it might look like:

[1] Yes, you will have 70 points even if you copy the solution on pages 3–5. However, what is the point in it for you? Try to implement it yourself! Or at least understand what the code is doing and why it is done like that.

```
public static void main(String[] args) {
        LinkedList myList = new LinkedList();
        myList.addLast(2);
        myList.addLast(4);
        myList.addLast(8);
        myList.addFirst(-2);
        myList.addFirst(-4);
        myList.addLast(9);
        myList.print();
        System.out.println(myList.indexOf(4));
        System.out.println(myList.contains(9));

        for (int i = 0; i < 6; ++i) {
                myList.deleteLast();
                myList.print();
        }
}
```

**Hint.** To build the LinkedList class, you need to implement a small private class Node inside the LinkedList. You may do it, for instance, like that:

```
private class Node {
        private int value;
        private Node next;

        public Node(int value) {
                this.value = value;
        }
}
```

Problem 2. [30 points if in $O(n)$, 25 points otherwise[2]] Add a new method reverse to the class LinkedList created in the problem 1. This method must reverse the order of the nodes in the list. For example, having applied reverse to the list $[2 \to 4 \to 8 \to 9 \to 8]$, we will change it to $[8 \to 9 \to 8 \to 4 \to 2]$.

[2] Here you are on your own. Yes, it is a challenge, but you can do it!

Possible solution to the problem 1.
Use it only for your reference, do not copy it. Try to implement it yourself first!

```
import java.util.NoSuchElementException;

public class LinkedList {
        private Node head; // first
        private Node tail; // last

        private class Node {
                private int value;
                private Node next;

                public Node(int value) {
                        this.value = value;
                }
        }

        private boolean hasNext(Node node) {
                return (node.next != null);
        }

        private boolean isEmpty() {
                return (head == null);
        }

        public void print() {
                Node current = head;
                System.out.print("[");

                while (current != null) {
                        if (hasNext(current)) {
                                System.out.print(current.value + ", ");
                        } else {
                                System.out.print(current.value);
                        }
                        current = current.next;
                }

                System.out.println("]");
        }

        public void addFirst(int value) {
                Node node = new Node(value);

                if (isEmpty()) {
                        head = tail = node;
                } else {
                        node.next = head;
                        head = node;
                }
        }
```

```java
public void addLast(int value) {
        Node node = new Node(value);
        if (isEmpty()) {
                head = tail = node;
        } else {
                tail.next = node;
                tail = node;
        }
}

public int indexOf(int value) {
        int index = 0;
        Node current = head;

        while (current != null) {
                if (current.value == value) {
                        return index;
                }
                index++;
                current = current.next;
        }

        return -1;
}

public boolean contains(int value) {
        return (indexOf(value) != -1);
}

public void deleteFirst() {
        if (isEmpty()) {
                throw new NoSuchElementException();
        }

        if (head == tail) {
                head = tail = null;
                return;
        }

        Node formerHeadNext = head.next;
        head.next = null;
        head = formerHeadNext;
}

public Node previous(Node node) {
        if (isEmpty())
                throw new NoSuchElementException();

        Node current = head;
        while (current.next != node) {
                if (!hasNext(current)) {
                        throw new NoSuchElementException();
                        }
                        current = current.next;
                }
                return current;

        }

        public void deleteLast() {
                if (isEmpty())
                        throw new NoSuchElementException();

                if (head == tail) {
                        head = tail = null;
                        return;
                }

                Node lastButOne = previous(tail);
                lastButOne.next = null;
                tail = lastButOne;
        }

}
```