**1. Print in reverse order: You are asked to design a method in linked list to print data in reverse order. You don't need to reverse linked list permanently.**

```
  void printRev(){
        Node current = head;
        Linked_List temp = new Linked_List();
        while (current!= null) {
            temp.addFront(current.data);

            current = current.next;
        }
        temp.printList();
    }



}
class Main{

    public static void main(String[] args) {
        Linked_List ll = new Linked_List();
        ll.addBack(1);
        ll.addBack(2);
        ll.addFront(3);
        ll.addBack(4);
        ll.addBack(5);
        ll.addFront(6);
        System.out.println("Print Original");
        ll.printList();
        System.out.println("Print Reverse");
        ll.printRev();
        System.out.println("Reverse Linked_List");
        ll.Reverse();
        ll.printList();
        System.out.println("Size: " + ll.getSize());
    }

```

```
Print Original
6 3 1 2 4 5
Print Reverse
5 4 2 1 3 6
Reverse Linked_List
5 4 2 1 3 6
Size: 6
azharali@fedora:~/Semester 3/DSA LAB/LAB5$
```

**2. Balanced Brackets: Take user string input and check whether it's balanced or not. Use stack functions. Input may contain any of the bracket among {, [, ( and any number and letters like: ({[a+b]+c}-1) and so on.**

```java
import java.util.Stack;
public class Balanced_Brackets {

    boolean Is_Banlaced( String s1 ) {
        Stack <Character> stack = new Stack<>();

        char c1 [] = s1.toCharArray();
        for(char k : c1){
            if(k == '(' || k == '{'  || k =='['){
                stack.push(k);
            }
            else if (k == ')' || k == '}'  || k ==']'){
                if(stack.isEmpty()){
                    return false;
                }
                char t = stack.pop();
                if((k ==')' && t != '(') || (k =='}' && t != '{') || (k ==']' && t != '[')){
                    return false;
                }
            }

        }

        return stack.isEmpty();
    }
}



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

        Balanced_Brackets b1 = new Balanced_Brackets();
        String input = "({[a+b]+c}-1)";

        boolean result = b1.Is_Banlaced(input);

        System.out.println("Is the input String Balanaced : "+result);


    }
}
```

```
Is the input String Balanaced : true
azharali@fedora:~/Semester 3/DSA LAB/LAB5$
```

**3. FirstSingleLetter: Create the function char firstSingleLetter (const char text [], const int n) which finds and returns the first letter of text that occurs only once. n is the number of characters in the text.**

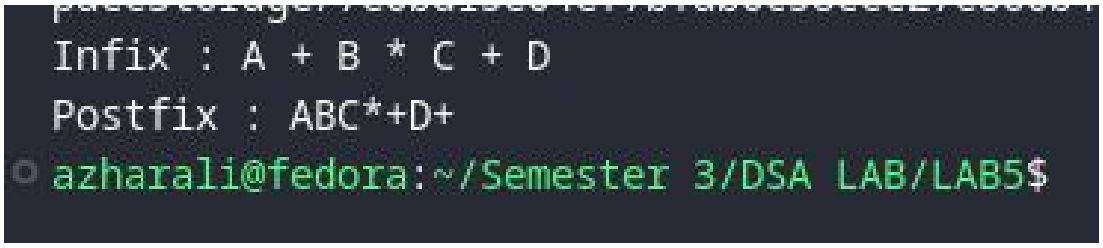
```java
public class FirstSingleLatter {

    char CheckFirstSingleLatter(char  text[] ){
        final char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();

        int count [] = new int [26];

        for (char c : text) {

            if (c >= 'a' && c <= 'z') {
                count[c - 'a']++;
            }
        }

        for (char c : text) {
            if (c >= 'a' && c <= 'z' && count[c - 'a'] == 1) {
                return c;
            }
        }
        return'\0';

    }

    public static void main(String[] args) {

        FirstSingleLatter f1 = new FirstSingleLatter();
        final char arr[] = {'a','l','g','o','r','i','t','h','m'};
        System.out.println(f1.CheckFirstSingleLatter(arr));

    }
}

```

```
a
azharali@fedora:~/Semester 3/DSA LAB/LAB5$
```

## 4. Convert Infix expression to Postfix expression using Stack data structure.

**Input: A + B * C + D** **Output: ABC*+D+**

```java
import java.util.Stack;

class InfixToPostfix {

    static int precedence(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
        }
        return -1;
    }

    static boolean isOperator(char x) {
        return (x == '+' || x == '-' || x == '*' || x == '/');
    }

    static String infixToPostfix(String inf_exp) {
        Stack<Character> stack = new Stack<>();
        String result = "";

        for (int i = 0; i < inf_exp.length(); i++) {
            char c = inf_exp.charAt(i);

            if (c == ' ')
                continue;

            if (Character.isLetterOrDigit(c)) {
                result += c;
            }

            else if (c == '(') {
                stack.push(c);
            }

            else if (c == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    result += stack.pop();
                }
                stack.pop();
            }

            else if (isOperator(c)) {
                while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
                    result += stack.pop();
                }
                stack.push(c);
            }
        }

        while (!stack.isEmpty()) {
            result += stack.pop();
        }

        return result;
    }

    public static void main(String args[]) {
        String infixExp = "A + B * C + D";
        System.out.println("Infix : " + infixExp);
        System.out.println("Postfix : " + infixToPostfix(infixExp));
    }
}
```


Infix : A + B * C + D
Postfix : ABC*+D+
azharali@fedora:~/Semester 3/DSA LAB/LAB5$

# THE END