Tree Calculate Expression

จัดทำโดย

นางสาวชลนิกานต์	**สังวัง**	**6530200070**
นางสาวณัฐณิชา	**นนตระอุดร**	**6530200134**
นางสาวสุภาวดี	**ม่วงประเสริฐ**	**6530200860**
นายจักรภพ	**ศิวะกุลรังสรรค์**	**6530200584**

หมู่เรียนบรรยาย 800 ปีการศึกษา 2566

ราบวิชา Data Structures and Algorithms รหัสวิชา 01418231

นำเสนอ

ผศ.ดร.จิรวรรณ เจริญสุข

คณะวิทยาศาสตร์ ศรีราชา สาขาวิทยาการคอมพิวเตอร์

มหาวิทยาลัยเกษตรศาสตร์ศรีราชา

โปรแกรมคำนวณค่า Prefix Postfix Infix สร้างขึ้นโดยใช้

https://www.onlinegdb.com

สามารถเข้าไปดูเพิ่มเติมและทดลองใช้งานได้ที่ : https://onlinegdb.com/L0DqF5qYy

Code

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.Stack;

class Node {

    String data; //ประกาศตัวแปรเพื่อเก็บข้อมูลใน node
    Node left, right; //Connection(left, right) ใช้สำหรับเก็บการอ้างอิงไปยัง nodeซ้ายและขวาของ node ปัจจุบัน

    public Node(String data) { //Constructorที่รับพารามิเตอร์ data
        this.data = data; //กำหนดค่าให้กับตัวแปร data
        left = right = null; //และกำหนดค่า null ให้กับ left และ right
    }
}

public class Main {

    public static boolean isOperator(String ch) { //เมธอด isOperator ที่ใช้เพื่อตรวจสอบว่าสตริง ch เป็น operator (+, -, *, /, ^) หรือไม่ และคืนค่า true และ false
        return (ch.equals("+") || ch.equals("-") || ch.equals("*") || ch.equals("/") || ch.equals("^"));
    }

    public static Node expressionTree(String[] equation, int notation) { //เมธอด expressionTree สร้าง tree จากสมการคณิตศาสตร์ (Prefix, Infix, หรือ Postfix)
        Stack<Node> st = new Stack<Node>(); //สร้าง Object Stack “st” เพื่อใช้เก็บ node ในการสร้าง tree
        switch (notation) { //คำสั่ง switch ใช้ค่า notation เพื่อกำหนดวิธีการแปลงสมการ
            case 1: //Prefix
```

```
            for (int i = equation.length - 1; i >= 0; i--) { //ลูป for เริ่มจาก equation ตัวสุดท้าย แล้ว
สร้าง tree ตามลำดับ
                st = PrePost_fix(equation[i], st); //เรียกใช้เมธอด PrePost_fix เพื่อแปลงและสร้าง node
ให้เป็น Prefix
            }
            break;
        case 2: //Infix
            Node t1,
             t2,
             temp,
             node;
            for (int i = 0; i < equation.length; i++) { //ลูป for เริ่มจาก equation ตัวแรก แล้วสร้าง tree
ตามลำดับ
                if (Arrays.asList(equation).contains("(") && Arrays.asList(equation).contains(")")) { //
ตรวจสอบว่าสมการมี “(” และ “)” หรือไม่
                    st = findbracket(equation[i], st); //ใช้เมธอด findbracket จัดการกับวงเล็บ
                } else {
                    if (!isOperator(equation[i])) { //ถ้า equation ไม่เป็น operetor
                        temp = new Node(equation[i]); //สร้าง Node ใหม่ “temp” และใส่ข้อมูลจาก
equation[i] เข้าไปใน temp
                        st.push(temp); //เก็บใน Stack “st”
                    } else {
                        temp = new Node(equation[i]);


                        t1 = st.pop(); //pop ข้อมูลสุดท้ายใน Stack “st” ออก
                        i++; //เลื่อนเพื่ออ่าน operator
                        node = new Node(equation[i]); //สร้าง Node ใหม่เพื่อเก็บ operator
                        st.push(node); // ใส่ node ใน Stack “st”
                        t2 = st.pop();
```

```
                    temp.left = t1;
                    temp.right = t2;

                    st.push(temp);
                }
            }
        }
        break;
    case 3: //Postfix
        for (int i = 0; i < equation.length; i++) { //ลูป for เริ่มจาก equation ตัวแรก แล้วสร้าง tree ตามลำดับ
            st = PrePost_fix(equation[i], st); //เรียกใช้เมธอด PrePost_fix เพื่อแปลงและสร้าง node ให้เป็น Postfix
        }
        break;
    default:
        return null;
    }
    return st.pop();
}

public static void display(Node root) { //เมธอด display ใช้แสดง Expression Tree ในรูปแบบ Inorder (left-root-right)
    if (root != null) { //if เพื่อตรวจสอบว่า root ไม่ใช่ node ว่าง
        display(root.left); //เรียกใช้เมธอด display ใน left child node
        System.out.print(root.data); //แสดง data ของ node ปัจจุบัน (operand)
        display(root.right);
    }
}
```

```
    public static Stack<Node> PrePost_fix(String equation, Stack<Node> st) { //เมธอด PrePost_fix ใช้สร้าง Expression tree ตามรูปแบบ Prefix หรือ Postfix
        Node t1, t2, temp;
        if (!isOperator(equation)) { //ถ้า equation ไม่ใช่ operator
            temp = new Node(equation); //สร้าง “temp” เก็บ equation ใน node นี้
            st.push(temp);
        } else {
            temp = new Node(equation);

            t1 = st.pop();
            t2 = st.pop();

            temp.left = t2;
            temp.right = t1;

            st.push(temp);
        }
        return st;
    }

    public static Stack<Node> findbracket(String equation, Stack<Node> st) { //เมธอด findbracket ที่ใช้ค้นหาและประมวลผลวงเล็บ
        int limit = 0; //ประกาศตัวแปร limit เพื่อเก็บลำดับ “)”
        Node t1, t2, temp, node;
        if (equation.equals(")")) { //ถ้า equation เป็น “)”
            for (int j = st.size() - 1; j >= 0; j--) { //ลูป for เริ่มจากตัวสุดท้ายของ Stack “st”
                node = st.get(j); //ดึง node ออกจาก Stack “st” ที่มีลำดับ j
                if (node.data.equals("(")) { //ถ้า equation เป็น “(”
                    limit = j; //เก็บลำดับไว้ใน limit
                    break;
```

```
                }
            }
            for (int k = st.size() - 1; k > limit; k--) { //ลูป for เริ่มจากตัวสุดท้ายของ Stack “st” ไปจนถึง limit
                node = st.get(k);
                if (isOperator(node.data) && node.left == null && node.right == null) { //ถ้า node นี้เป็น operator และ ทางซ้ายกับขวาเป็น null
                    st.remove(node); //ลบ node จาก Stack “st”
                    temp = new Node(node.data); //สร้าง node “temp” เพื่อเก็บ operator จาก node ก่อนหน้า
                    t1 = st.pop(); //pop node ออกจาก Stack “st” และเก็บใน t1
                    t2 = st.pop();
                    temp.left = t2;
                    temp.right = t1;
                    if (st.peek().data.equals("(")) { //ถ้า node ที่อยู่ด้านบนสุดของ Stack “st” เป็น “(”
                        st.pop();
                    }
                    st.push(temp);
                    k--; //ลดค่า k เพื่อตรวจสอบ node ถัดไปใน loop
                }
            }
        } else { //ถ้า equation ไม่ใช่ “)”
            temp = new Node(equation); //สร้าง node “temp” เก็บ equation
            st.push(temp); //เก็บ temp ใน Stack “st”
        }
        return st;
    }

    public static int calculateTree(Node Tree) { //เมธอด calculateTree ใช้คำนวณผลลัพธ์ของ Expression Tree
```

```
        if (Tree == null) {
            return 0;
        } else if (Tree.left == null && Tree.right == null) { //ถ้า node “Tree” ไม่มี left child, right child (Tree เป็น leaf node เก็บค่า operand)
            return Integer.parseInt(Tree.data); //แปลงข้อมูลเป็น int
        }
        int left = calculateTree(Tree.left); //เรียกเมธอด calculateTree แบบ recursive บน left child ของ Tree และเก็บไว้ใน left
        int right = calculateTree(Tree.right);
        if (Tree.data.equals("+")) { //ถ้า operator ใน Tree คือ “+”
            return left + right;
        } else if (Tree.data.equals("-")) {
            return left - right;
        } else if (Tree.data.equals("*")) {
            return left * right;
        } else {
            return left / right;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int Select;
        String input;
        String[] equation;
        String[] choice = {"Prefix", "Infix", "Postfix"};
        System.out.println("Tree Calculate Expression");
        for (int i = 0; i < choice.length; i++) {//ลูป for แสดงเมนูเลือกวิธีการแปลงสมการ (Prefix, Infix, Postfix, และ Exit)
            System.out.printf("%d.%s\n", i + 1, choice[i]);
```

```
        }
        System.out.println("4.Exit program");
        System.out.print("Your choice number is : ");
        Select = sc.nextInt();
        sc.nextLine();
        if (Select > 0 && Select <= 3) {
            System.out.printf("Your %s : ", choice[Select - 1]);
            input = sc.nextLine();
            equation = input.split(" "); //แยกสมการโดยใช้ช่องว่างเป็นตัวแยกและเก็บใน equation
            Node r = expressionTree(equation, Select); //เรียกเมธอด expressionTree เพื่อสร้าง
Expression Tree จากสมการทางคณิตศาสตร์ที่ input เข้ามา และเก็บ Expression Tree ที่สร้างใน “r”
            System.out.print("Display at inorder : "); //แสดงผลลัพธ์การคำนวณจาก Expression Tree ใน
ลำดับ Inorder
            display(r); //เรียกเมธอด display เพื่อแสดงผลลัพธ์การคำนวณจาก Expression Tree ในลำดับ
Inorder
            System.out.println("\nsum of equations : " + calculateTree(r));
        } else if (Select == 4) {
        } else {
            System.out.println("Sorry,we don't have that choice in fuction");
        }
    }
}
```

Terminal

```
Tree Calculate Expression
1.Prefix
2.Infix
3.Postfix
4.Exit program
Your choice number is :
```

หน้าจอรับค่าข้อมูลจากแป้นพิมพ์

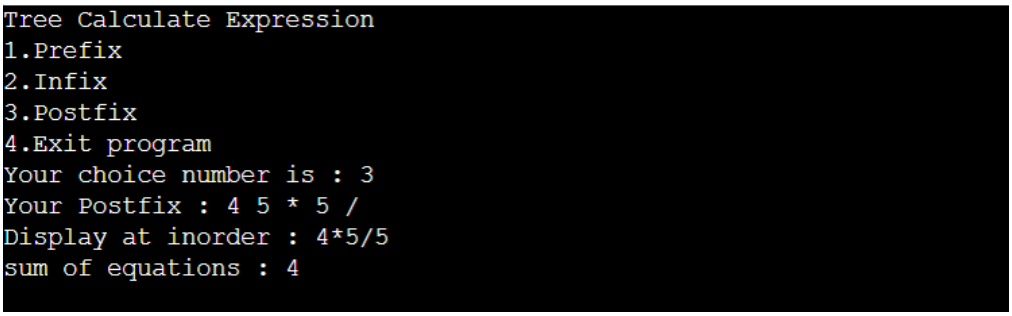
```
Tree Calculate Expression
1.Prefix
2.Infix
3.Postfix
4.Exit program
Your choice number is : 1
Your Prefix : + 3 + 4 5
Display at inorder : 5+4+3
sum of equations : 12
```

เลือก choice ที่ต้องการ

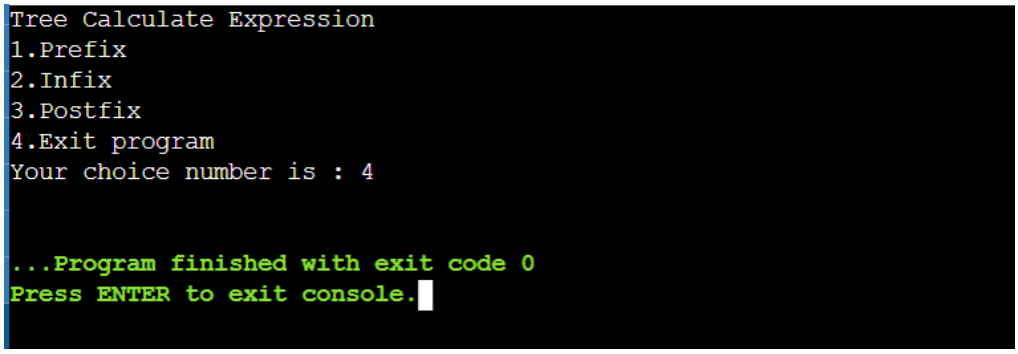
กรณีเลือก 1 ต้องการหาผลรวม Prefix : ใส่เลข Prefix ที่ต้องการคำนวนลงไป --> จะได้ผลรวมตามในรูป

```
Tree Calculate Expression
1.Prefix
2.Infix
3.Postfix
4.Exit program
Your choice number is : 2
Your Infix : ( 3 - ( 4 - 5 ) )
Display at inorder : 3-4-5
sum of equations : 4
```

กรณีเลือก 2 ต้องการหาผลรวม Infix : ใส่เลข Infix ต้องการคำนวนลงไป --> จะได้ผลรวมตามในรูป



กรณีเลือก 3 ต้องการหาผลรวม Postfix : ใส่เลข Postfix ต้องการคำนวนลงไป --> จะได้ผลรวมตามในรูป



กรณีเลือก 4 ต้องการออกจากโปรแกรม โปรแกรมจะหยุดทำงานและได้ผลลัพธ์ดังรูป