LAPORAN PRAKTIKUM
STRUKTUR DATA NON LINIER
MODUL 6

Dosen Pengampu

JB. Budi Darmawan S.T., M.Sc.

DISUSUN OLEH

Yohanis Celvin D. P. U. Pati

245314033

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

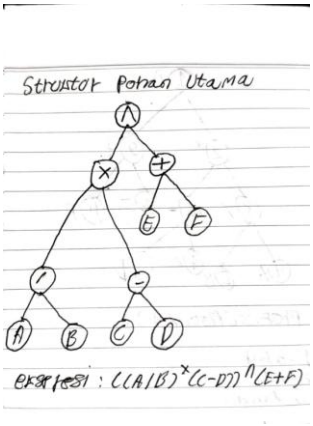
UNIVERSITAS SANATA DHARMA

YOGYAKARTA

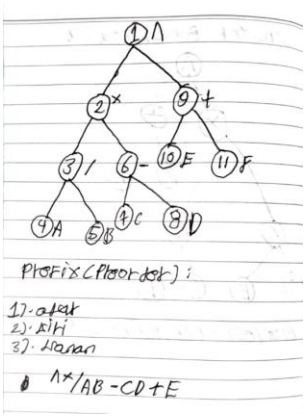
2025

A. Ilustrasi pohon biner prefix, infix dan postfix

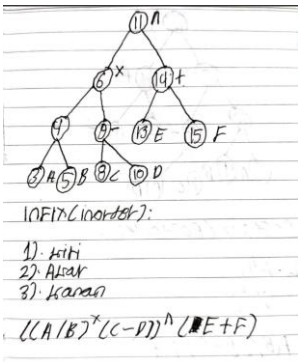
- **Struktur dasar**



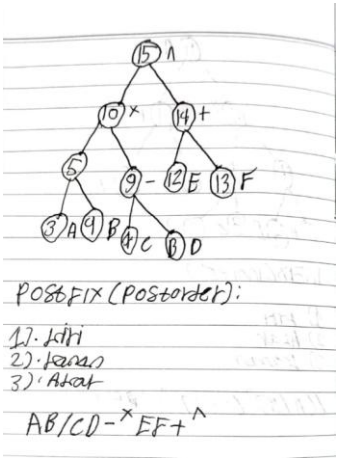
- **Prefix (Preorder)**



- **Infix (Inorder)**



- **Postfix (Postorder)**



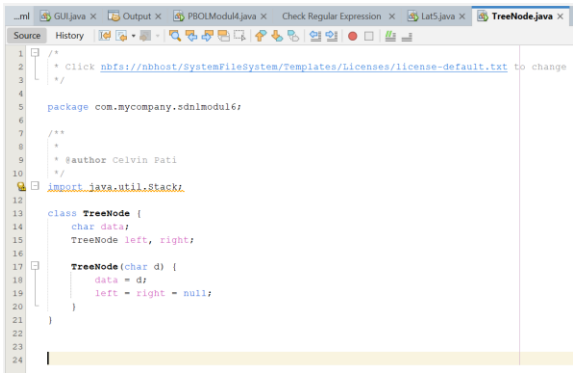
B. DIAGRAM UML

TreeNode
data : char **left : TreeNode** **right : TreeNode**
TreeNode(char d)

Tree
root : TreeNode
addExpressionInfix(x:String) : void **printPrefix() : void** **printInfix() : void** **printPostfix() : void**

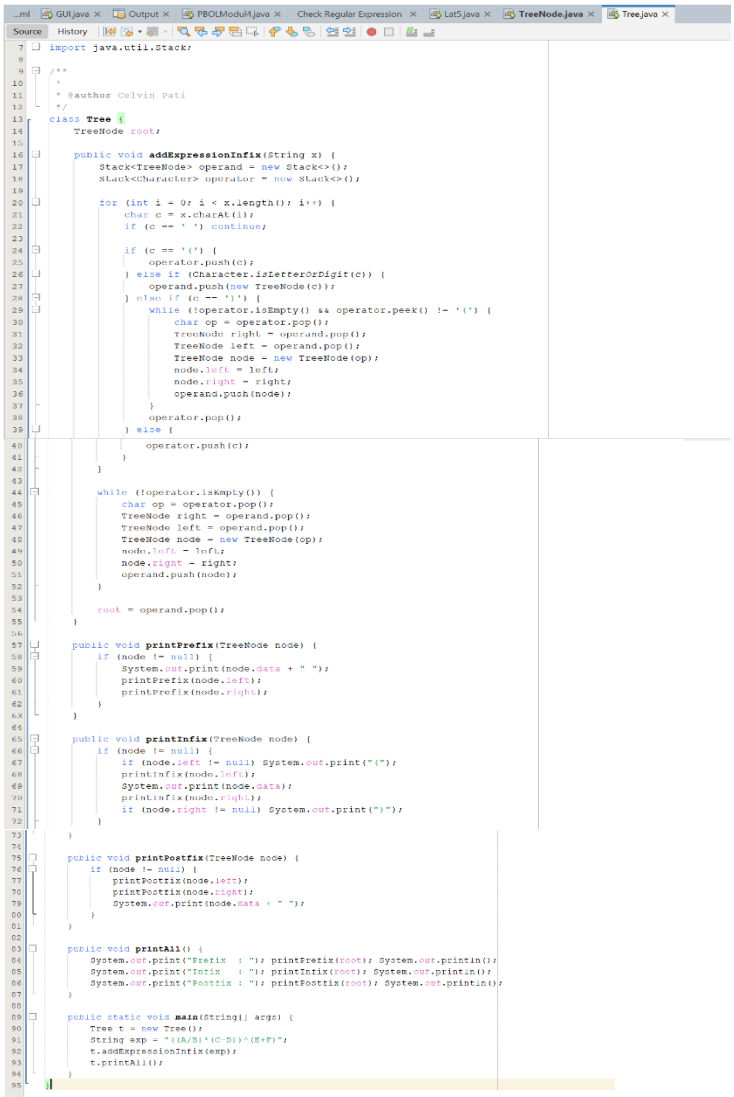
C. SOURCE CODE

- **TreeNode**



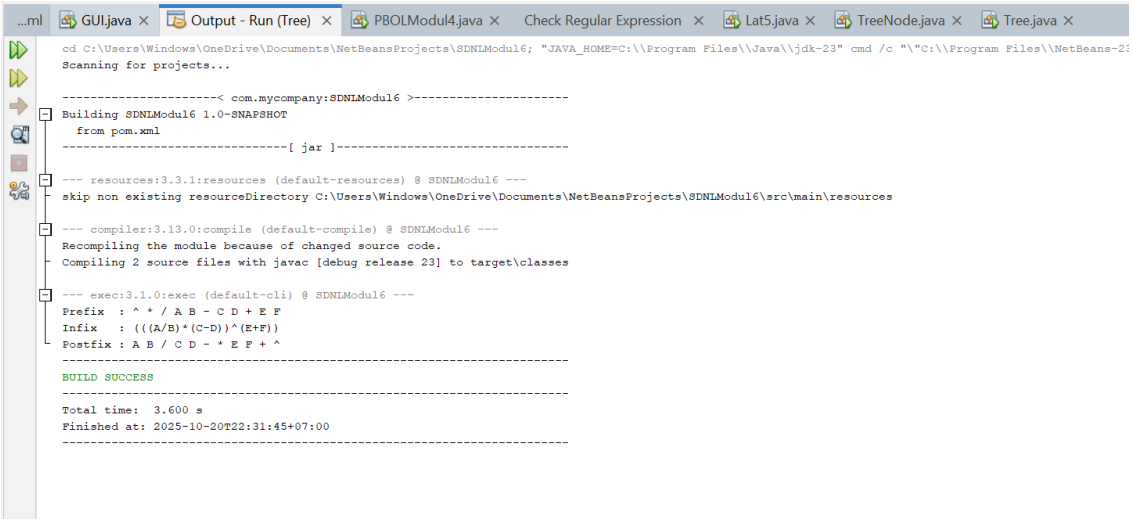
```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
 */

package com.mycompany.sdnlmodul6;

/**
 *
 * @author Celvin Pati
 */
import java.util.Stack;

class TreeNode {
    char data;
    TreeNode left, right;

    TreeNode(char d) {
        data = d;
        left = right = null;
    }
}
```

- **Tree**



```
import java.util.Stack;

/**
 *
 * @author Celvin Pati
 */
class Tree {
    TreeNode root;

    public void addExpressionInfix(String x) {
        Stack<TreeNode> operand = new Stack<>();
        Stack<Character> operator = new Stack<>();

        for (int i = 0; i < x.length(); i++) {
            char c = x.charAt(i);
            if (c == ' ') continue;

            if (c == '(') {
                operator.push(c);
            } else if (Character.isLetterOrDigit(c)) {
                operand.push(new TreeNode(c));
            } else if (c == ')') {
                while (!operator.isEmpty() && operator.peek() != '(') {
                    char op = operator.pop();
                    TreeNode right = operand.pop();
                    TreeNode left = operand.pop();
                    TreeNode node = new TreeNode(op);
                    node.left = left;
                    node.right = right;
                    operand.push(node);
                }
                operator.pop();
            } else {
                operator.push(c);
            }
        }

        while (!operator.isEmpty()) {
            char op = operator.pop();
            TreeNode right = operand.pop();
            TreeNode left = operand.pop();
            TreeNode node = new TreeNode(op);
            node.left = left;
            node.right = right;
            operand.push(node);
        }

        root = operand.pop();
    }

    public void printPrefix(TreeNode node) {
        if (node != null) {
            System.out.print(node.data + " ");
            printPrefix(node.left);
            printPrefix(node.right);
        }
    }

    public void printInfix(TreeNode node) {
        if (node != null) {
            if (node.left != null) System.out.print("(");
            printInfix(node.left);
            System.out.print(node.data);
            printInfix(node.right);
            if (node.right != null) System.out.print(")");
        }
    }

    public void printPostfix(TreeNode node) {
        if (node != null) {
            printPostfix(node.left);
            printPostfix(node.right);
            System.out.print(node.data + " ");
        }
    }

    public void printAll() {
        System.out.print("Prefix  : "); printPrefix(root); System.out.println();
        System.out.print("Infix   : "); printInfix(root); System.out.println();
        System.out.print("Postfix : "); printPostfix(root); System.out.println();
    }

    public static void main(String[] args) {
        Tree t = new Tree();
        String exp = "((A/B)*(C-D))^(E+F)";
        t.addExpressionInfix(exp);
        t.printAll();
    }
}
```

D. OUTPUT



Penjelasan:

1. **Prefix : ^ * / A B - C D + E F**

   Ini hasil dari preorder traversal (akar - kiri - kanan) yang artinya, setiap operator dituliskan **sebelum** operand-operandnya, dengan urutan Langkah kunjungan:

   - ^ (akar utama)
   - turun ke kiri *
   - kiri lagi /
   - A, lalu B
   - kembali ke kanan –
   - C, D
   - ke kanan akar +
   - E, F

   Dan dengan output itu berarti traversal preorder saya sudah berfungsi dengan baik.

2. **Infix : ((A / B) * (C - D)) ^ (E + F)**

   Ini adalah hasil dari inorder traversal (kiri – akar – kanan). Dan ekspresi ini adalah bentuk asli infix yang gampang dibaca secara sistematis. Dan ada tanda kurung untuk memastikan urutan operasi:

   - (A / B) dilakukan dulu
   - hasilnya dikali (C - D)
   - hasil seluruhnya dipangkatkan dengan (E + F)

   dan ini sesuai dengan aturan operator precendence.

3. **Postfix : A B / C D - * E F + ^**

   Hasil dari posorder traversal (kiri – kana – akar). Yang berarti setiap operator dituliskan sesudah operand-operandnya, postfix bisa dipakai untuk evaluasi dengan stack, karena tidak harus perlu tanda kurung, Langkah-langkaahnya:

   - A B / hasil pertama
   - C D - hasil kedua
   - * mengalikan dua hasil sebelumnya
   - E F + hasil ketiga
   - ^ mempangkatkan hasil pertama dengan ketiga

   Berarti traversal posorder juga bekerja dengan baik.

E. ANALISA

Berikut Analisa saya permethod calss program saya TreeNode dan Tree:

1. **TreeNode**
   a. char data untuk menyimpan karakter dari elemen pohon (bisa operand seperti A, B, atau operator seperti +, *, ^).
   b. TreeNode left, right sebagai referensi ke simpul anak kiri dan anak kanan (struktur pohon biner).
   c. Konstruktor TreeNode(char d) adalah inisialisai simpul baru dengan data d dan menetapkan anak kiri & kanan ke null. Yang berarti saat simpul dibuat pertama kali, belum memiliki cabang.
2. **Tree**
   a. **Atribut TreeNode root** ini menunjuk pada akar (root) dari pohon ekspresi yang akan dibentuk. Awalnya null.
   b. Method 1 addExpressionInfix(String x)
      - Stack<TreeNode> operand untuk menyimpan simpul operand (A, B, C...) atau hasil subpohon.
      - Stack<Character> operator untuk menyimpan tanda operator (+, -, *, /, ^).
      - Loop for untuk membaca karakter per karakter dari string ekspresi infix.
      - Jika '(' Didorong (push) ke stack operator berarti tanda awal subekspresi.

- Jika **operand** (A–Z atau angka) dibuat simpul baru TreeNode(c) lalu didorong ke stack operand.
- Jika **')'`** ini tanda akhir subekspresi, maka dilakukan pop: ambil 2 operand (kanan & kiri) dan 1 operator bentuk pohon kecil push kembali ke stack operand.
- Jika **operator** (+, -, *, /, ^) dimasukkan ke stack operator untuk nanti diolah saat ada) atau akhir string.
- Setelah loop selesai semua operator tersisa diproses dengan cara yang sama akan membentuk satu pohon penuh.
- root = operand.pop(); node terakhir di stack operand menjadi akar dari seluruh pohon ekspresi.

c. Method 2 printPrefix(TreeNode node)

- Fungsi untuk menampilkan notasi prefix (preorder traversal).
- Proses:
  - Cetak data pada node saat ini (akar).
  - Rekursif ke subtree kiri.
  - Rekursif ke subtree kanan.
- **Hasil ^ * / A B - C D + E F**

d. Method 3 printInfix(TreeNode node)

- Fungsi untuk menampilkan notasiinfix (inorder traversal)
- Proses:
  - Buka kurung sebelum anak kiri.
  - Cetak node (operator).
  - Tutup kurung setelah anak kanan.
- **Hasil ((A/B)*(C-D))^(E+F)**

e. Method 4 printPostfix(TreeNode node)

- Fungsi untuk menampilkan notasi postfix (postorder traversal).
- Proses:
  - Rekursif ke kiri
  - Lalu kanan
  - Terakhir cetak node
- **Hasil A B / C D - * E F + ^**

f. Method 5 printAll()

- Fungsi untuk mencetak seluruh hasil traversal dalam satu eksekusi.
- Proses memanggil printPrefix, printInfix, dan printPostfix secara berurutan dengan labelnya masing-masing.
- Output:

  Prefix : ^ * / A B - C D + E F

  Infix : ((A/B)*(C-D))^(E+F)

  Postfix : A B / C D - * E F + ^

g. Method Main

- Fungsi sebagai entry point untuk menjalankan program.
- Proses:
  - Membuat objek Tree.
  - Menetapkan ekspresi infix.
  - Memanggil addExpressionInfix() untuk membangun pohon.
  - Menampilkan traversal hasilnya.

F. REFERENSI