Laporan Praktikum

Struktur Data Non Linear DP

Modul 5

Dosen Pengampu

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

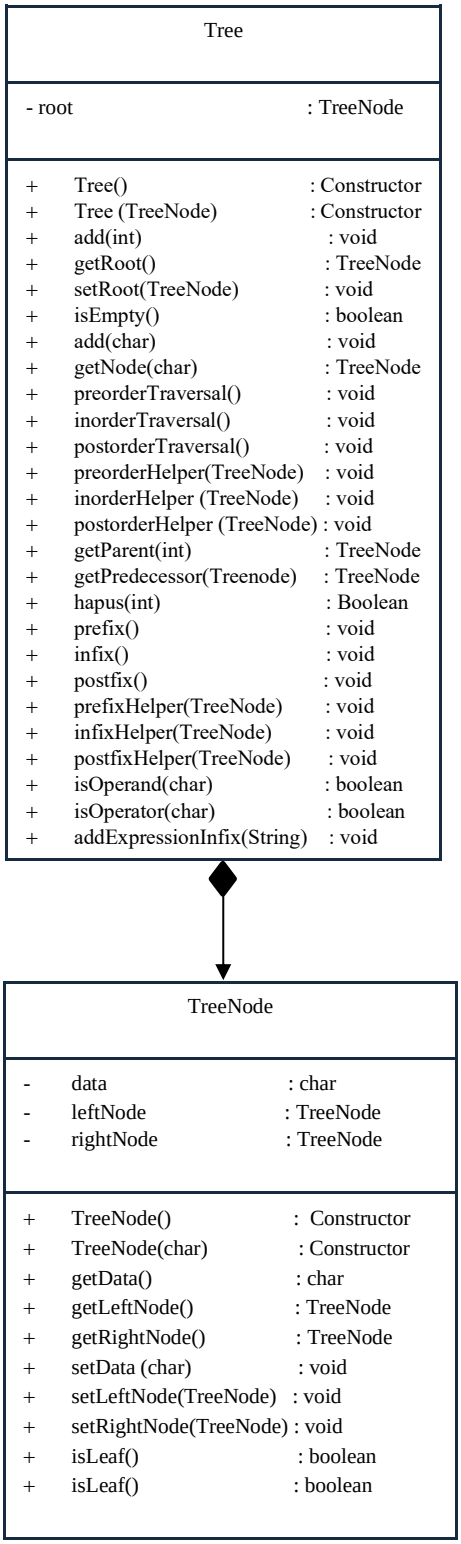
Oleh :

Nama : Maria Gresia Plena Br Purba

NIM : 235314094

PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS SANATA DHARMA
YOGYAKARTA
2024

A. Diagram UML



B. Source Code

TreeNode

```java
package modul5;

public class TreeNode {

    char data;
    TreeNode leftNode;
    TreeNode rightNode;

    public TreeNode() {
    }

    public TreeNode(char data) {
        this.data = data;
        leftNode = null;
        rightNode = null;
    }

    public char getData() {
        return data;
    }

    public TreeNode getLeftNode() {
        return leftNode;
    }

    public TreeNode getRightNode() {
        return rightNode;
    }

    public void setData(char data) {
        this.data = data;
    }

    public void setLeftNode(TreeNode leftNode) {
        this.leftNode = leftNode;
    }

    public void setRightNode(TreeNode rightNode) {
        this.rightNode = rightNode;
    }

    public boolean isLeaf() {
        if (leftNode == null && rightNode == null) {
            return true;
        } else {
            return false;
        }
    }
}
```

Tree

```
216     public void prefix() {
217         prefixHelper(localRoot: root);
218         System.out.println();
219     }
220
221     public void infix() {
222         infixHelper(localRoot: root);
223         System.out.println();
224     }
225
226     public void postfix() {
227         postfixHelper(localRoot: root);
228         System.out.println();
229     }
230
231     public void prefixHelper(TreeNode localRoot) {
232         if (localRoot != null) {
233             System.out.print(localRoot.getData() + " ");
234             prefixHelper(localRoot: localRoot.getLeftNode());
235             prefixHelper(localRoot: localRoot.getRightNode());
236         }
237     }
238
239     public void infixHelper(TreeNode localRoot) {
240         if (localRoot != null) {
241             infixHelper(localRoot: localRoot.getLeftNode());
242             System.out.print(localRoot.getData() + " ");
243             infixHelper(localRoot: localRoot.getRightNode());
244         }
245     }
246
247     public void postfixHelper(TreeNode localRoot) {
248         if (localRoot != null) {
249             postfixHelper(localRoot: localRoot.getLeftNode());
250             postfixHelper(localRoot: localRoot.getRightNode());
251             System.out.print(localRoot.getData() + " ");
252         }
253     }
```

```
254
255     public boolean isOperand(char a) {
256         String operand ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
257
258         for (int i = 0; i < operand.length(); i++) {
259             if (a == operand.charAt(index: i)) {
260                 return true;
261             }
262         }
263         return false;
264     }
265
266     public boolean isOperator(char a) {
267         return a == '*'|| a == '/' || a == '+' || a == '-' || a == '%';
268     }
269
270     public void addExpressionInfix(String x) {
271         Stack<TreeNode> Operand = new Stack();
272         Stack<TreeNode> Operator = new Stack();
273
274         char c;
275
276         for (int i = 0; i < x.length(); i++) {
277             c = x.charAt(index: i);
278
279             if (c == '(') {
280                 Operator.push(new TreeNode(data: c));
281             } else if (isOperand(a: c)) {
282                 Operand.push(new TreeNode(data: c));
283             } else if (isOperator(a: c)) {
284                 while (!Operator.isEmpty() && Operator.peek().getData() != '(') {
285                     TreeNode operator = Operator.pop();
286                     TreeNode rightOperand = Operand.pop();
287                     TreeNode leftOperand = Operand.pop();
288
289                     operator.setRightNode(rightNode: rightOperand);
290                     operator.setLeftNode(leftNode: leftOperand);
291
292                     Operand.push(item: operator);
```

```
                    Operand.push(item: operator);
                }
                Operator.push(new TreeNode(data: c));

            } else if (c == ')') {
                while (!Operator.isEmpty() && Operator.peek().getData() != '(') {
                    TreeNode operator = Operator.pop();
                    TreeNode rightOperand = Operand.pop();
                    TreeNode leftOperand = Operand.pop();

                    operator.setRightNode(rightNode: rightOperand);
                    operator.setLeftNode(leftNode: leftOperand);

                    Operand.push(item: operator);
                }
                Operator.pop();
            }
        }
        while (!Operator.isEmpty()) {
            TreeNode operator = Operator.pop();
            TreeNode rightOperand = Operand.pop();
            TreeNode leftOperand = Operand.pop();

            operator.setRightNode(rightNode: rightOperand);
            operator.setLeftNode(leftNode: leftOperand);

            Operand.push(item: operator);
        }
        if (!Operand.isEmpty()) {
            root = Operand.pop();
        }
    }
}
```

TreeMain

```
package modul5;

public class TreeMain {

    public static void main(String[] args) {
        Tree expressionTree = new Tree();

        String expression = "(A+B)";
        expressionTree.addExpressionInfix(x: expression);

        System.out.println(x: "Prefix : ");
        expressionTree.prefix();

        System.out.println(x: "Infix : ");
        expressionTree.infix();

        System.out.println(x: "Postfix : ");
        expressionTree.postfix();
    }
}
```

C. Output

```
run:
Prefix :
+ A B
Infix :
A + B
Postfix :
A B +
BUILD SUCCESSFUL (total time: 0 seconds)
```

D. Analisa

1. Penjelasan singkat mengenai kegunaan method-method yang dibuat pada praktikum modul 5

TreeNode

Tree

Code	Penjelasan

public void prefix() {
prefixHelper(root);
System.out.println();
}
	Mengembalikan nilai dari prefixHelper()

public void infix() {
infixHelper(root);
System.out.println();
}
	Mengembalikan nilai dari infixHelper ()

public void postfix() {
postfixHelper(root);
System.out.println();
}
	Mengembalikan nilai dari postfixHelper ()

public void infixHelper(TreeNode localRoot) {
if (localRoot != null) {
infixHelper(localRoot.getLeftNode());
System.out.print(localRoot.getData() + "
");
infixHelper(localRoot.getRightNode());
}
}
	Memeriksa apakah localRoot bernilai null atau tidak, jika tidak maka akan mengambil data dari localRoot dan mencetak node kiri dan kanan dari localRoot dengan memanggil method prefixHelper.

public void infixHelper(TreeNode localRoot) {
if (localRoot != null) {
infixHelper(localRoot.getLeftNode());
System.out.print(localRoot.getData() + "
");
infixHelper(localRoot.getRightNode());
}
}
	Memeriksa apakah localRoot bernilai null atau tidak, jika tidak maka akan mencetak node kiri dari localRoot dengan memanggil method infixHelper kemudian, mengambil data dari localRoot dan mencetak node kanan dari localRoot dengan memanggil method infixHelper.

public void postfixHelper(TreeNode localRoot)
{
if (localRoot != null) {
postfixHelper(localRoot.getLeftNode());
 	
postfixHelper(localRoot.getRightNode());
System.out.print(localRoot.getData() + "
");
	Memeriksa apakah localRoot bernilai null atau tidak, jika tidak maka akan mencetak node kiri dan kanan dari localRoot dengan memanggil method postfixHelper. Kemudian,

} }	mengambil data dari localRoot.

public boolean isOperand(char a) {
String operand ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
 	
for (int i = 0; i < operand.length(); i++) {
if (a == operand.charAt(i)) {
return true;
}
}
return false;
}
	Memeriksa apakah karakter a termasuk operand atau tidak dengan membandingkannya dengan huruf yang ada pada String operand.

public boolean isOperator(char a) {
return a == '*'|| a == '/' || a == '+' || a == '-' || a == '%';
}
	Memeriksa apakah data termasuk operator yang bertipe char dengan mengembalikan nilai dari a, yaitu '*' atau '/' atau '+' atau '-' atau '%'

public void addExpressionInfix(String x) {
Stack<TreeNode> Operand = new Stack();
Stack<TreeNode> Operator = new Stack();
 	
char c;
 	
for (int i = 0; i < x.length(); i++) {
c = x.charAt(i);
 	
if (c == '(') {
Operator.push(new TreeNode(c));
} else if (isOperand(c)) {
Operand.push(new TreeNode(c));
} else if (isOperator(c)) {
while (!Operator.isEmpty() && Operator.peek().getData() != '(') {
TreeNode operator = Operator.pop();
TreeNode rightOperand = Operand.pop();
TreeNode leftOperand = Operand.pop();
 	
operator.setRightNode(rightOperand);
operator.setLeftNode(leftOperand);
 	
Operand.push(operator);
}
Operator.push(new TreeNode(c));
	Melakukan perulangan for selama nilai i kurang dari panjang x.
	
	Jika nilai dari c sama dengan "(", maka "(" akan di-push ke operator
	
	Jika c adalah operand, maka c akan di-push ke operand
	
	Jika c adalah operator, maka selama Operator tidak kosong dan tidak bernilai "(" maka akan menyimpan nilai Operator yang di pop ke dalam operator, menyimpan nilai dari rightOperand dan leftOperand dengan Operand yang di pop. Mengeset nilai dari operator bagian kanan dengan rightOperand dan kiri dengan leftOperand. Lalu, mem-push nilai dari operator c ke Operator.

} else if (c == ')') {
while (!Operator.isEmpty() &&
Operator.peek().getData() != '(') {
TreeNode operator =
Operator.pop();
TreeNode rightOperand =
Operand.pop();
TreeNode leftOperand =
Operand.pop();
 	
operator.setRightNode(rightOperand);
operator.setLeftNode(leftOperand);
 	
Operand.push(operator);
}
Operator.pop();
}
}
while (!Operator.isEmpty()) {
TreeNode operator = Operator.pop();
TreeNode rightOperand =
Operand.pop();
TreeNode leftOperand = Operand.pop();
 	
operator.setRightNode(rightOperand);
operator.setLeftNode(leftOperand);
 	
Operand.push(operator);
}
if (!Operand.isEmpty()) {
root = Operand.pop();
}
}
	Jika c sama dengan “)”, maka selama Operator tidak kosong dan tidak bernilai “(“, maka akan menyimpan nilai Operator yang di pop ke dalam operator, menyimpan nilai dari rightOperand dan leftOperand dengan Operand yang di pop. Mengeset nilai dari operator bagian kanan dengan rightOperand dan kiri dengan leftOperand. Lalu, mem-push nilai dari operator c ke Operand. Lalu, “(“ akan di-pop dari Operator.
	
	Selama Operator tidak kosong maka akan menyimpan nilai Operator yang di pop ke dalam operator, menyimpan nilai dari rightOperand dan leftOperand dengan Operand yang di pop. Mengeset nilai dari operator bagian kanan dengan rightOperand dan kiri dengan leftOperand. Lalu, mem-push nilai dari operator ke Operand.
	
	Jika setelah proses, Operand tidak kosong, maka Operand akan di pop dan akan menjadi root

TreeNode

Code	Penjelasan
private char data;	Mendeklarasikan atribut data dengan tipe data char yang bersifat private.
private TreeNode leftNode;	Mendeklarasikan atribut leftNode dengan tipe data TreeNode yang bersifat private.
private TreeNode rightNode;	Mendeklarasikan atribut rightNode dengan tipe data TreeNode yang bersifat private.

public TreeNode() {
this(0);
}
	Berfungsi untuk memanggil constructor yang lain. Lalu, menginisialisasikan data dengan nilai 0.

public TreeNode(char data) {
this.data = data;
leftNode = null;
rightNode = null;
}
	Menginisialisasi data dengan nilai yang ada. Menginisialisasi leftNode, serta rightNode dengan null.

public char getData() {
return data;
}
	Method ini berfungsi untuk mengembalikan nilai dari data.

public TreeNode getLeftNode() {
return leftNode;
}
	Method in berfungsi untuk mengembalikan nilai dari leftNode.

public TreeNode getRightNode() {
return rightNode;
}
	Method in berfungsi untuk mengembalikan nilai dari rightNode.

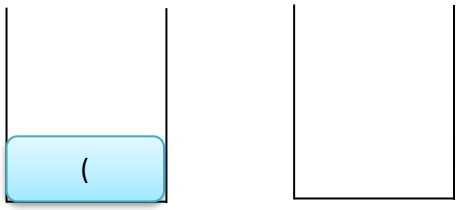
public void setData(char data) {
this.data = data;
}
	Method ini berfungsi untuk mengubah nilai data saat ini dengan data baru yang ingin diinput.

public void setLeftNode(TreeNode lefftNode) {
this.leftNode = leftNode;
}
	Method ini berfungsi untuk mengubah nilai leftNode saat ini dengan leftNode baru yang ingin diinput.

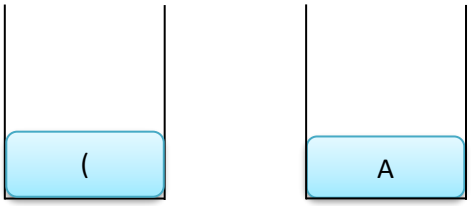
public void setRightNode(TreeNode rightNode) {
this.rightNode = rightNode;
}
	Method ini berfungsi untuk mengubah nilai rightNode saat ini dengan rightNode baru yang ingin diinput.

public boolean isLeaf() {
if (leftNode == null && rightNode == null) {
return true;
} else {
return false;
}
}
	Method ini berfungsi untuk mengetahui apakah suatu node memiliki anak atau tidak dengan syarat jika leftNode dan rightNode bernilai null.

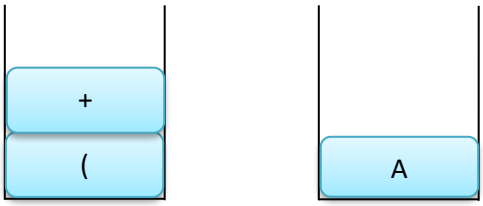
2. **Ilustrasi addExpressionInfix menggunakan stack dan tree, gunakan inputan parameternya (A+B)**



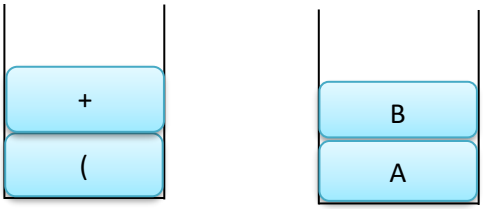
Operator **Operand**



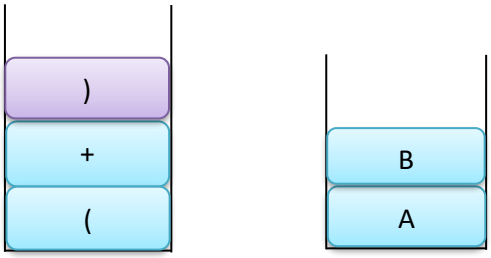
Operator **Operand**



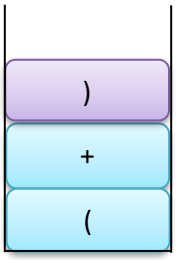
Operator **Operand**



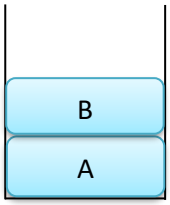
Operator **Operand**



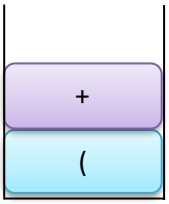
Operator **Operand**



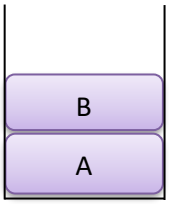
Operator



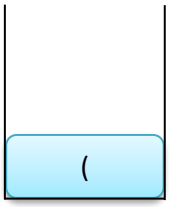
Operand



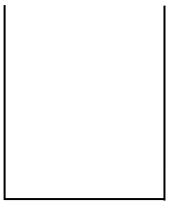
Operator



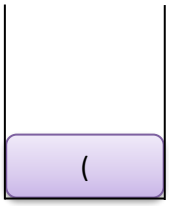
Operand



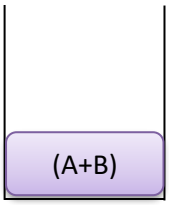
Operator



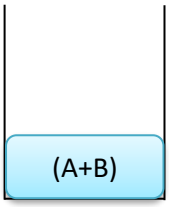
Operand



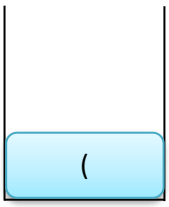
Operator



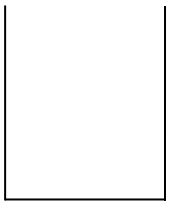
Operand



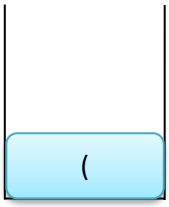
3. **Ilustrasi cara cetak infix saja dengan menggunakan tree yang terbentuk dari poin 2**



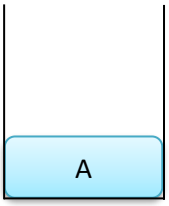
Operator



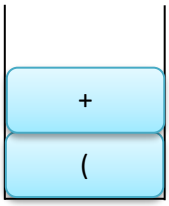
Operand



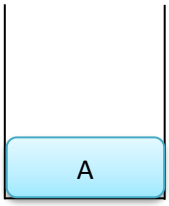
Operator



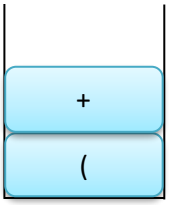
Operand



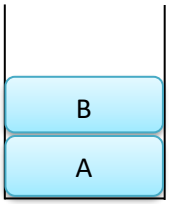
Operator



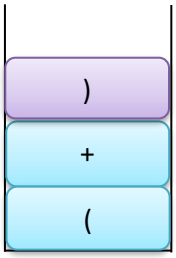
Operand



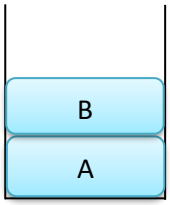
Operator



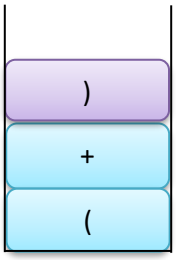
Operand



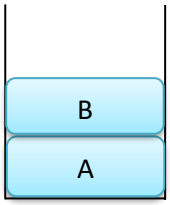
Operator



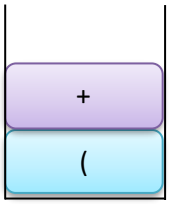
Operand



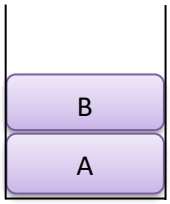
Operator



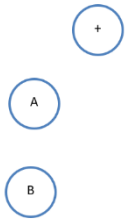
Operand

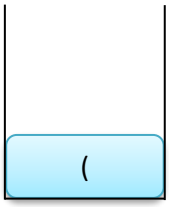


Operator

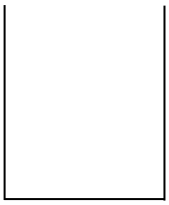


Operand

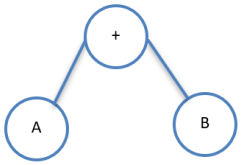


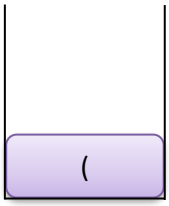


Operator

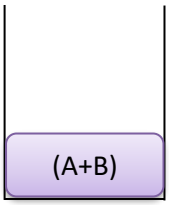


Operand





Operator



Operand