# 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

	Tree	
- root		: TreeNode
	Tree()	: Constructor
	Tree (TreeNode)	: Constructor
	add(int)	: void
	getRoot()	: TreeNode
	setRoot(TreeNode)	: void
	isEmpty()	: boolean
	add(char)	: void
	getNode(char)	: TreeNode
,	preorderTraversal()	: void
	inorderTraversal()	: void
	postorderTraversal()	: void
	preorderHelper(TreeNode)	
	inorderHelper (TreeNode)	: void
	postorderHelper (TreeNode	/
	getParent(int)	: TreeNode
+	getPredecessor(Treenode)	: TreeNode
+ 1	hapus(int)	: Boolean
+ 1	prefix()	: void
+ :	infix()	: void
+ 1	postfix()	: void
+ j	prefixHelper(TreeNode)	: void
+	infixHelper(TreeNode)	: void
+ 1	postfixHelper(TreeNode)	: void
+	isOperand(char)	: boolean
	isOperator(char)	: boolean
	addExpressionInfix(String)	: void



TreeNode				
- - -	data leftNode rightNode	: char : TreeNode : TreeNode		
+	TreeNode()	: Constructor		
+	TreeNode(char)	: Constructor		
+	getData()	: char		
+	getLeftNode()	: TreeNode		
+	getRightNode()	: TreeNode		
+	setData (char)	: void		
+	setLeftNode(TreeNode)	: void		
+	setRightNode(TreeNode	e): void		
+	isLeaf()	: boolean		
+	isLeaf()	: boolean		

## **B. Source Code**

#### TreeNode

```
package modul5;
 2
 3
       public class TreeNode {
 4
          char data;
 6
          TreeNode leftNode;
 7
          TreeNode rightNode;
 8
 9
          public TreeNode() {
10
11
12 -
          public TreeNode (char data) {
              this.data = data;
13
              leftNode = null;
14
              rightNode = null;
15
16
17
18 -
          public char getData() {
19
             return data;
20
21
22 -
          public TreeNode getLeftNode() {
23
             return leftNode;
24
25
26 -
          public TreeNode getRightNode() {
27
            return rightNode;
28
29
30
   public void setData(char data) {
              this.data = data;
31
32
33
34
          public void setLeftNode (TreeNode leftNode) {
35
              this.leftNode = leftNode;
36
37
   public void setRightNode(TreeNode rightNode) {
38
39
              this.rightNode = rightNode;
40
41
   public boolean isLeaf() {
42
              if (leftNode == null && rightNode == null) {
44
                  return true;
45
                  return false;
46
47
48
49
```

#### Tree

```
216 _
             public void prefix() {
                 prefixHelper(localRoot: root);
217
218
                  System.out.println();
219
220
221 _
             public void infix() {
                 infixHelper(localRoot: root);
222
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) {
                 if (localRoot != null) {
232
                     System.out.print(localRoot.getData() + " ");
233
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) {
                 String operand ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
256
257
                 for (int i = 0; i < operand.length(); i++) {
259
                    if (a == operand.charAt(index:i)) {
                        return true;
260
261
262
263
                 return false;
264
265
266
    public boolean isOperator(char a) {
                 return a == '*'|| a == '/' || a == '+' || a == '-' || a == '%';
267
268
269
270
    public void addExpressionInfix(String x) {
271
                 Stack<TreeNode> Operand = new Stack();
Stack<TreeNode> Operator = new Stack();
272
273
274
                 char c;
275
276
                 for (int i = 0; i < x.length(); i++) {
277
                     c = x.charAt(index:i);
278
                     if (c == '(') {
279
                        Operator.push(new TreeNode(data: C));
280
                     } 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() != '(') {
                             TreeNode operator = Operator.pop();
285
                             TreeNode rightOperand = Operand.pop();
286
287
                             TreeNode leftOperand = Operand.pop();
288
289
                             operator.setRightNode(rightNode: rightOperand);
290
                             operator.setLeftNode(leftNode: leftOperand);
291
                             Operand.push(item: operator);
```

```
293
                              Operand.push(item: operator);
294
                          Operator.push(new TreeNode(data: C));
295
296
                      } else if (c == ')') {
297
                          while (!Operator.isEmpty() && Operator.peek().getData() != '(') {
298
299
                             TreeNode operator = Operator.pop();
                              TreeNode rightOperand = Operand.pop();
300
                              TreeNode leftOperand = Operand.pop();
301
                              operator.setRightNode(rightNode: rightOperand);
304
                              operator.setLeftNode(leftNode:leftOperand);
305
306
                              Operand.push(item: operator);
307
308
                          Operator.pop();
309
310
                  while (!Operator.isEmpty()) {
311
312
                     TreeNode operator = Operator.pop();
                      TreeNode rightOperand = Operand.pop();
313
314
                     TreeNode leftOperand = Operand.pop();
 315
316
                      operator.setRightNode(rightNode: rightOperand);
317
                      operator.setLeftNode(leftNode:leftOperand);
318
319
                      Operand.push(item: operator);
320
321
                  if (!Operand.isEmpty()) {
                      root = Operand.pop();
322
323
324
325
```

#### **TreeMain**

```
3
      public class TreeMain {
 5 📮
          public static void main(String[] args) {
             Tree expressionTree = new Tree();
              String expression = "(A+B)";
              expressionTree.addExpressionInfix(x: expression);
10
11
              System.out.println(x: "Prefix : ");
              expressionTree.prefix();
12
13
              System.out.println(x: "Infix : ");
14
15
              expressionTree.infix();
16
              System.out.println(x: "Postfix : ");
17
              expressionTree.postfix();
18
19
```

# 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 ${\bf 5}$

## TreeNode

#### Tree

Code	Penjelasan
public void prefix() {	Mengembalikan nilai dari
<pre>prefixHelper(root);</pre>	prefixHelper()
System.out.println();	
}	
public void infix() {	Mengembalikan nilai dari
infixHelper(root);	infixHelper ()
System.out.println();	
muhlio void mostfiv() [	Manaamhalikan nilai dani
<pre>public void postfix() {     postfixHelper(root);</pre>	Mengembalikan nilai dari postfixHelper ()
System.out.println();	postrixiteiper ()
System.out.printin(),	
public void infixHelper(TreeNode localRoot) {	Memeriksa apakah
if (localRoot != null) {	localRoot bernilai null atau
infixHelper(localRoot.getLeftNode());	tidak, jika tidak maka akan
System.out.print(localRoot.getData() + "	mengambil data dari
");	localRoot dan mencetak
infixHelper(localRoot.getRightNode());	node kiri dan kanan dari
}	localRoot dengan
}	memanggil method
	prefixHelper.
<pre>public void infixHelper(TreeNode localRoot) {</pre>	Memeriksa apakah
if (localRoot != null) {	localRoot bernilai null atau
<pre>infixHelper(localRoot.getLeftNode());</pre>	tidak, jika tidak maka akan
System.out.print(localRoot.getData() + "	mencetak node kiri dari
");	localRoot dengan
infixHelper(localRoot.getRightNode());	memanggil method
}	infixHelper kemudian,
}	mengambil data dari
	localRoot dan mencetak
	node kanan dari localRoot
	dengan memanggil method
nublic void poetfix Halper(TreeNode localPoot)	infixHelper.  Memeriksa apakah
public void postfixHelper(TreeNode localRoot)	localRoot bernilai null atau
if (localRoot != null) {	tidak, jika tidak maka akan
postfixHelper(localRoot.getLeftNode());	mencetak node kiri dan
postriatierper(localitoot.get.erit vouc()),	kanan dari localRoot
<pre>postfixHelper(localRoot.getRightNode());</pre>	dengan memanggil method
System.out.print(localRoot.getData() + "	postfixHelper. Kemudian,
");	

```
mengambil data dari
                                                 localRoot.
public boolean isOperand(char a) {
                                                 Memeriksa apakah karakter
     String operand
                                                 a termasuk operand atau
="ABCDEFGHIJKLMNOPQRSTUVWXYZab
                                                 tidak dengan
cdefghijklmnopqrstuvwxyz";
                                                 membandingkannya dengan
                                                 huruf yang ada pada String
     for (int i = 0; i < operand.length(); i++) {
                                                 operand.
       if (a == operand.charAt(i)) {
         return true;
     }
     return false;
public boolean isOperator(char a) {
                                                 Memeriksa apakah data
     return a == '*'|| a == '/' || a == '+' || a == '-' ||
                                                 termasuk operator yang
a == '%':
                                                 bertipe char dengan
                                                 mengembalikan nilai dari a,
  }
                                                 yaitu '*' atau '/' atau '+'
                                                 atau '-' atau '%'
public void addExpressionInfix(String x) {
                                                 Melakukan perulangan for
     Stack<TreeNode> Operand = new Stack():
                                                 selama nilai i kurang dari
     Stack<TreeNode> Operator = new Stack();
                                                 panjang x.
                                                 Jika nilai dari c sama
     char c;
                                                 dengan "(", maka "(" akan
     for (int i = 0; i < x.length(); i++) {
                                                 di-push ke operator
       c = x.charAt(i);
                                                 Jika c adalah operand,
       if (c == '(') \{
                                                 maka c akan di-push ke
          Operator.push(new TreeNode(c));
                                                 operand
       } else if (isOperand(c)) {
         Operand.push(new TreeNode(c));
                                                 Jika c adalah operator,
       } else if (isOperator(c)) {
                                                 maka selama Operator tidak
         while (!Operator.isEmpty() &&
                                                 kosong dan tidak bernilai
Operator.peek().getData() != '(') {
                                                 "(" maka akan menyimpan
            TreeNode operator =
                                                 nilai Operator yang di pop
                                                 ke dalam operator,
Operator.pop();
            TreeNode rightOperand =
                                                 menyimpan nilai dari
                                                 rightOperand dan
Operand.pop();
            TreeNode leftOperand =
                                                 leftOperand dengan
                                                 Operand vang di pop.
Operand.pop();
                                                 Mengeset nilai dari
                                                 operator bagian kanan
                                                 dengan rightOperand dan
operator.setRightNode(rightOperand);
            operator.setLeftNode(leftOperand);
                                                 kiri dengan leftOperand.
                                                 Lalu, mem-push nilai dari
                                                 operator c ke Operator.
            Operand.push(operator);
          Operator.push(new TreeNode(c));
```

```
\} 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.

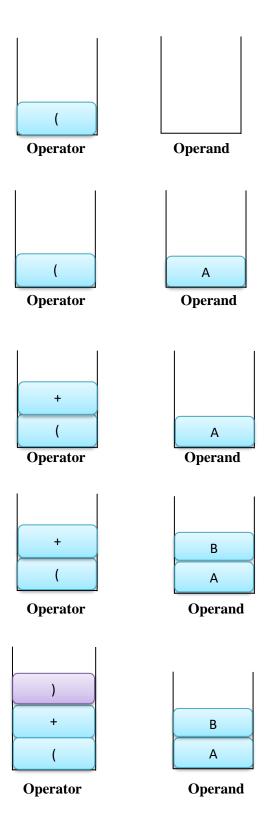
Selama Operator tidak kosong maka akan menyimpan nilai Operator yang di pop ke dalam operator, menyimpan nilai dari rightOperand dan leftOperand dengan Operand yang di pop. Mengeset nilai dari operator bagian kanan dengan rightOperand dan kiri dengan leftOperand. Lalu, mem-push nilai dari operator ke Operand.

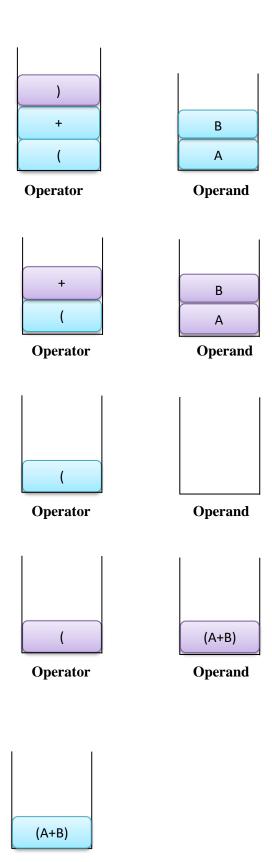
Jika setelah proses, Operand tidak kosong, maka Operand akan di pop dan akan menjadi root

# 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.
<pre>public TreeNode() {</pre>	Berfungsi untuk memanggil
this(0);	constructor yang lain. Lalu,
}	menginisialisasikan data dengan nilai 0.
<pre>public TreeNode(char data) {</pre>	Menginisialisasi data dengan nilai yang
this.data = data;	ada. Menginisialisasi leftNode, serta
leftNode = null;	rightNode dengan null.
rightNode = null;	
}	
<pre>public char getData() {</pre>	Method ini berfungsi untuk
return data;	mengembalikan nilai dari data.
}	
<pre>public TreeNode getLeftNode() {</pre>	Method in berfungsi untuk
return leftNode;	mengembalikan nilai dari leftNode.
}	
<pre>public TreeNode getRightNode() {</pre>	Method in berfungsi untuk
return rightNode;	mengembalikan nilai dari rightNode.
}	
<pre>public void setData(char data) {</pre>	Method ini berfungsi untuk mengubah
this.data = data;	nilai data saat ini dengan data baru yang
}	ingin diinput.
public void setLeftNode(TreeNode	Method ini berfungsi untuk mengubah
lefftNode) {	nilai leftNode saat ini dengan leftNode
this.leftNode = leftNode;	baru yang ingin diinput.
}	
public void setRightNode(TreeNode	Method ini berfungsi untuk mengubah
rightNode) {	nilai rightNode saat ini dengan
this.rightNode = rightNode;	rightNode baru yang ingin diinput.
muhlia haalaan (-1 f) (	Mathadini hanfungai antalanan (1.1.
public boolean isLeaf() {	Method ini berfungsi untuk mengetahui
if (leftNode == null &&	apakah suatu node memiliki anak atau
rightNode == null) {	tidak dengan syarat jika leftNode dan
return true;	rightNode bernilai null.
} else { return false;	
Teturn raise,	
, }	
J	

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





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

