

NAMA : CHANDRA HARKAT RAHARJA
NPM : 233040089
KELAS : PRAKTIKUM PEMROGRAMAN I (B)
GITHUB:[ComradeChandra/PP12025_B_233040089](https://github.com/ComradeChandra/PP12025_B_233040089)

LATIHAN-1

Latihan 1

Latihan ini sudah dilakukan dipertemuan sebelumnya yaitu membuat kelas Node sebagai representasi dari elemen Node List. Berikut kode program kelas Node menggunakan bahasa Java

Pseudocode	Bahasa Pemrograman
<pre>class Node { data: tipe data (T) next: Node }</pre>	<pre>public class Node { private int data; private Node next; /** Inisialisasi atribut node */ public Node(int data) { this.data = data; } /** Setter & Getter */ }</pre>

Penjelasan dan Kode:

```
TugasDanPertemuanCoding > TugasPertemuan4 > J Node.java >
1 public class Node {
2
3     private int data;
4     private Node next;
5
6
7
8     // Inisialisasi atribut node //
9     public Node(int data){
10         this.data = data;
11     }
12
13
14
15     // Getter dan Setter //
16     public int getData() {
17         return data;
18     }
19
20     public void setData(int data) {
21         this.data = data;
22     }
23
24     public Node getNext() {
25         return next;
26     }
27
28     public void setNext(Node next) {
29         this.next = next;
30     }
31 }
32 }
```

Deklarasi Kelas dan Atribut:

- ❖ `public class Node` mendefinisikan kelas Node yang dapat diakses dari kelas lain.
- ❖ `private int data` menyimpan nilai data pada node.
- ❖ `private Node next` menyimpan referensi ke node selanjutnya, memungkinkan pembuatan linked list.

Konstruktor:

- ❖ `public Node(int data)` menginisialisasi objek Node dengan nilai data yang diberikan.
- ❖ Di dalam konstruktor, `this.data = data;` menetapkan nilai parameter ke atribut data.
- ❖ Atribut `next` tidak diinisialisasi, sehingga secara default bernilai null.

Getter dan Setter:

- ❖ `getData()` dan `setData(int data)` digunakan untuk mengakses dan mengubah nilai data.
- ❖ `getNext()` dan `setNext(Node next)` digunakan untuk mengakses dan mengatur referensi ke node berikutnya.

LATIHAN-2

Latihan 2

Latihan ini akan memberikan implementasi operasi penambahan/sisipan elemen list di tengah/*middle* dengan notasi algoritma. Operasi ini direpresentasikan dengan fungsi **addMid** dengan parameter data yaitu node dan indeks yang akan ditambahkan ke List.

- Tambahkan fungsi dibawah ini di kelas **StrukturList**. Fungsi addMid di bawah dikonversi ke dalam bahasa pemrograman

Algoritma Fungsi addMid

```
procedure addMid(data: integer, position: integer)
deklarasi
    posNode, curNode: Node {current node}
    i: integer
deskripsi
    newNode ← new Node(data)
    IF (HEAD = null) THEN
        HEAD ← newNode
    ELSE
        curNode ← HEAD;
        IF (position = 1) THEN      {tambah di awal}
            newNode.next ← curNode
            HEAD ← newNode
            {slide berikutnya}
```

```
ELSE
    i ← 1
    WHILE (curNode <> null AND i < position) DO
        posNode ← curNode
        curNode ← curNode.next
        i++
    ENDWHILE
    posNode.next ← newNode
    newNode.next ← curNode
ENDIF
ENDIF
```

Penjelasan Dan Kode:

TugasDanPertemuanCoding > TugasPertemuan4 > StrukturList.java > StrukturList > addHead(int)

```
1 public class StrukturList {
2
3     Node HEAD;
4
5     public boolean isEmpty(){
6         return (HEAD == null);
7     }
8
9     //Menampilkan List//
10    public void display(){
11        Node curNode = HEAD;
12        while (curNode != null){
13            System.out.print(curNode.getData() + " ");
14            curNode = curNode.getNext();
15        }
16    }
17 }
18
19
20 //Prosedur untuk add head//
21
22 public void addHead(int data){
23     Node newNode = new Node(data);
24     if (isEmpty()){
25         HEAD = newNode;
26     }else{
27         newNode.setNext(HEAD);
28         HEAD = newNode;
29     }
30 }
```

(A). ADD HEAD

1.Deklarasi Kelas dan Variabel

❖ **public class StrukturList** adalah kelas yang digunakan untuk merepresentasikan linked list.

❖ Variabel **Node HEAD** berfungsi sebagai referensi ke node pertama dalam list.

2. Fungsi isEmpty()

❖ **public boolean isEmpty()** digunakan untuk mengecek apakah list kosong.

❖ Jika **HEAD == null**, maka fungsi akan mengembalikan **true**, yang berarti list kosong.

❖ Jika terdapat elemen dalam list, fungsi akan mengembalikan **false**.

3. Fungsi `display()`

- ❖ `public void display()` berfungsi untuk menampilkan semua elemen dalam list.
- ❖ Menggunakan variabel `curNode` untuk menelusuri list, dimulai dari `HEAD`.
- ❖ Selama `curNode` tidak `null`, program akan mencetak data dari node tersebut, kemudian pindah ke node berikutnya.

4. Fungsi `addHead(int data)`

- ❖ `public void addHead(int data)` digunakan untuk menambahkan elemen di awal list.
- ❖ Pertama, dibuat node baru dengan `new Node(data)`.
- ❖ Jika list kosong (`isEmpty() == true`), maka node baru langsung menjadi `HEAD`.
- ❖ Jika list sudah memiliki elemen, `newNode` akan menunjuk ke `HEAD` yang lama, lalu `HEAD` diperbarui ke `newNode`, sehingga node baru menjadi elemen pertama dalam list.

(B). ADD MIDDLE

```
// Prosedur untuk add mid berdasarkan algoritma
public void addMid(int data, int position) {
    Node newNode = new Node(data);

    if (HEAD == null) {
        // Jika list kosong, tambahkan node baru sebagai HEAD
        HEAD = newNode;
    } else {
        Node curNode = HEAD;
        if (position == 1) {
            // Tambahkan di awal
            newNode.setNext(curNode);
            HEAD = newNode;
        } else {
            Node posNode = null;
            int i = 1;

            // Iterasi hingga posisi yang diinginkan
            while (curNode != null && i < position) {
                posNode = curNode;
                curNode = curNode.getNext();
                i++;
            }

            // Sisipkan node baru di posisi yang ditentukan
            posNode.setNext(newNode);
            newNode.setNext(curNode);
        }
    }
}
```

1. Deklarasi dan Pembuatan Node Baru:

- ❖ Di awal prosedur, dibuat sebuah node baru menggunakan `Node newNode = new Node(data);`. Node ini akan menjadi elemen yang akan disisipkan ke dalam list.

2. Kondisi List Kosong:

- ❖ Prosedur mengecek apakah list kosong dengan memeriksa `if (HEAD == null)`. Jika list kosong, maka node baru langsung diassign ke `HEAD`, sehingga list mulai dari node tersebut.

3. Penambahan di Awal List (Posisi = 1):

- ❖ Jika list tidak kosong, langkah selanjutnya memeriksa apakah nilai `position` sama dengan 1. jika benar, node baru dimasukkan di awal list seperti pada metode `addHead: newNode.setNext(curNode);` mengarahkan node baru ke node saat ini (`HEAD`). `HEAD = newNode;` memperbarui `HEAD` sehingga node baru menjadi elemen pertama.

4. Penambahan di Tengah atau pada Posisi Tertentu (Posisi > 1):

- ❖ Jika `position` lebih dari 1, prosedur melakukan iterasi untuk menemukan posisi penyisipan: Variabel `curNode` diinisiasi dengan `HEAD` dan variabel `posNode` diinisiasi dengan `null` untuk menyimpan referensi node sebelumnya. Variabel penghitung `i` diinisiasi dengan nilai 1.

Proses Iterasi:

- Menggunakan perulangan `while (curNode != null && i < position)` untuk menelusuri list hingga mencapai posisi yang diinginkan atau mencapai akhir list.
- Di setiap iterasi, `posNode` diupdate dengan `curNode` sebagai node yang sedang diproses, lalu `curNode` berpindah ke node berikutnya dengan `curNode.getNext()`. Nilai `i` juga diincrement.

❖ Menyisipkan Node Baru:

Setelah iterasi selesai, `posNode.setNext(newNode)`; menghubungkan node sebelumnya ke node baru. Kemudian `newNode.setNext(curNode)`; mengarahkan node baru ke node yang tadinya berada di posisi tersebut (atau `null` jika iterasi selesai karena mencapai akhir list).

(C). ADD TAIL

```
//Prosedur untuk add tail//
public void addTail(int data){
    Node posNode = null, curNode = null;
    Node newNode = new Node(data);
    if (isEmpty()){
        HEAD = newNode;
    }else{
        curNode = HEAD;
        while (curNode != null){
            posNode = curNode;
            curNode = curNode.getNext();
        }
        posNode.setNext(newNode);
    }
}
```

1. Deklarasi dan Pembuatan Node Baru:

❖ Prosedur diawali dengan pembuatan node baru, yaitu `Node newNode = new Node(data)`; Node baru ini merupakan elemen yang nantinya akan disisipkan sebagai elemen terakhir (tail) dalam linked list.

2. Kondisi List Kosong:

❖ Prosedur memeriksa apakah list kosong dengan memanggil metode `isEmpty()`. Jika list kosong (artinya `HEAD == null`), maka node baru langsung dijadikan `HEAD`. Dengan begitu, pada list kosong node baru tersebut sekaligus menjadi elemen pertama maupun terakhir.

3. Menambahkan Node pada Posisi Tail (List Tidak Kosong):

❖ Jika list tidak kosong, variabel `curNode`

diinisialisasi dengan `HEAD` untuk memulai penelusuran list. Variabel `posNode` dideklarasikan untuk menyimpan referensi ke node terakhir yang telah dikunjungi selama iterasi.

Proses Iterasi:

❖ Perulangan dilakukan dengan `while (curNode != null)` yang menelusuri setiap node dalam list. Di setiap iterasi, `posNode` diperbarui dengan nilai `curNode`, kemudian `curNode` berpindah ke node berikutnya menggunakan `curNode.getNext()`. Setelah perulangan selesai, `posNode` mengacu pada node terakhir (tail lama) dalam list.

4. Menyisipkan Node Baru:

❖ Setelah menemukan tail lama, node baru disisipkan dengan cara menghubungkan node lama ke node baru melalui pemanggilan `posNode.setNext(newNode)`;.
Dengan cara ini, node baru berhasil ditambahkan sebagai elemen terakhir pada list.

LATIHAN-3

Latihan 3

Latihan ini akan memberikan penggunaan operasi penambahan elemen list (*head*, *tail* dan *middle*) dan kemudian menampilkan setiap elemen yang terdapat di list. Buatlah kelas **StrukturListTest** berikut fungsi main untuk mengeksekusi program. Konversikan urutan instruksi berikut di bawah ini ke fungsi tersebut!

Urutan Instruksi	Output
1. Create list dengan keyword new 2. Tambah elemen 3 di akhir list 3. Tambah elemen 4 di akhir list 4. Tambah elemen 7 di index 2 5. Tambah elemen 8 di index 2 6. Tambah elemen 5 di awal list 7. Tampilkan elemen list	5 3 8 7 4

Jawaban, Penjelasan dan Code:

```
J StrukturList.java M J Node.java J StrukturListTest.java X
TugasDanPertemuanCoding > TugasPertemuan4 > J StrukturListTest.java > ...
1 public class StrukturListTest {
2
3     Run | Debug
4     public static void main(String[] args) {
5         StrukturList list = new StrukturList();
6
7         // Tambah elemen 3 di akhir list
8         list.addTail(data:3);
9
10        // Tambah elemen 4 di akhir list
11        list.addTail(data:4);
12
13        // Tambah elemen 7 di index 2 (posisi tengah)
14        list.addMid(data:7, position:2);
15
16        // Tambah elemen 8 di index 2 (posisi tengah)
17        list.addMid(data:8, position:2);
18
19        // Tambah elemen 5 di awal list
20        list.addHead(data:5);
21
22        // Tampilkan elemen list
23        list.display(); // Output: 5 3 8 7 4
24    }
25 }
```

1. Deklarasi dan Pembuatan Objek List:

❖ Prosedur diawali dengan pembuatan objek list baru, yaitu `StrukturList list = new StrukturList();`. Objek ini merupakan representasi dari linked list yang nantinya akan dioperasikan. Pada saat ini, list masih kosong karena properti `HEAD` bernilai `null`.

2. Menambahkan Elemen di Akhir List (`addTail`):

❖ Operasi `list.addTail(3);`:
Prosedur dimulai dengan pembuatan node baru yang berisi data **3**. Karena list kosong (hasil pengecekan `isEmpty()` adalah `true`), node baru langsung dijadikan `HEAD`, sehingga list menjadi: **3**.

❖ Operasi `list.addTail(4);`:

Prosedur kembali membuat node baru dengan data **4**. Karena list tidak kosong, variabel `curNode` diinisialisasi dengan `HEAD` dan dilakukan iterasi untuk menelusuri node hingga mencapai node terakhir. Setelah iterasi selesai, node baru berisi **4** disisipkan setelah node terakhir menggunakan `posNode.setNext(newNode);`, menghasilkan list: **3, 4**.

3. Menambahkan Elemen di Posisi Tengah (`addMid`):

❖ Operasi `list.addMid(7, 2);`:

Prosedur membuat node baru dengan data **7**. Karena list tidak kosong dan posisi yang diminta adalah ke-2, dilakukan iterasi mulai dari `HEAD` hingga mencapai posisi yang ditentukan. Node baru dengan data **7** kemudian disisipkan di antara node yang sudah ada, sehingga list menjadi: **3, 7, 4**.

❖ Operasi `list.addMid(8, 2);`:

Prosedur kembali membuat node baru dengan data **8** dan menyisipkannya pada posisi ke-2. Dengan cara yang sama, iterasi dilakukan dari **HEAD**, dan node baru dengan data **8** disisipkan di depan node yang sebelumnya berada di posisi ke-2 (data 7), sehingga list berubah menjadi: **3, 8, 7, 4**.

4. Menambahkan Elemen di Awal List (`addHead`):

Prosedur `addHead` membuat node baru dengan data **5** menggunakan `new Node(5);`. Node baru ini disisipkan di awal list dengan cara mengarahkan node baru tersebut untuk menunjuk ke node yang sebelumnya merupakan **HEAD** (yaitu node dengan data 3), dan kemudian **HEAD** diperbarui sehingga node baru dengan data **5** menjadi elemen pertama, menghasilkan list akhir: **5, 3, 8, 7, 4**.

5. Menampilkan Elemen List:

Prosedur `display()` menelusuri list mulai dari **HEAD** hingga akhir dengan menggunakan perulangan. Setiap data node dicetak secara berurutan sehingga menghasilkan output: **5 3 8 7 4**

```
PS D:\KuliahSMT4\Praktikum Pemrograman 1> & 'C:\Program Files\Java\jre1.8.0_431\bin\java.exe' '-cp' 'C:\Users\ACER\AppData\Roaming\Code\User\workspaceStorage\47c3537e1c32a9f2a6c832819e9c66a8\redhat.java\jdt_ws\Praktikum Pemrograman 1_d3fc7d1e\bin' 'StrukturListTest'
5 3 8 7 4
PS D:\KuliahSMT4\Praktikum Pemrograman 1> ^C
PS D:\KuliahSMT4\Praktikum Pemrograman 1>
PS D:\KuliahSMT4\Praktikum Pemrograman 1> d::; cd 'd:\KuliahSMT4\Praktikum Pemrograman 1'; & 'C:\Program Files\Java\jre1.8.0_431\bin\java.exe' '-cp' 'C:\Users\ACER\AppData\Roaming\Code\User\workspaceStorage\47c3537e1c32a9f2a6c832819e9c66a8\redhat.java\jdt_ws\Praktikum Pemrograman 1_d3fc7d1e\bin' 'StrukturListTest'
5 3 8 7 4
PS D:\KuliahSMT4\Praktikum Pemrograman 1> 
```

TUGAS:

Tugas

1. Buatlah Struktur list untuk menambahkan data /node di awal, menengah dan akhir dengan tipe data valuenya adalah bilangan pecahan!
2. Lakukan pengujian terhadap operasi tersebut seperti pada latihan 3 sehingga membentuk deret bilangan seperti dibawah ini:
 - a. 2.1 3.4 4.5
 - b. 3.4 2.1 1.1 4.5 5.5

Catatan:

Laporan yang dikumpulkan berisi kode program dan hasil eksekusi

JAWAB:

A. Tambahan Code di File Node:

```
//tugas
// Tambahan node untuk data bertipe double
class NodeDouble {
    private double data;
    private NodeDouble next;

    public NodeDouble(double data) {
        this.data = data;
    }

    public double getData() {
        return data;
    }

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

    public NodeDouble getNext() {
        return next;
    }

    public void setNext(NodeDouble next) {
        this.next = next;
    }
}
```

Penjelasan:

Class **NodeDouble**

❖ Class **NodeDouble** dibuat sebagai node khusus untuk menyimpan data pecahan (**double**).

Strukturnya mirip dengan **Node** biasa, hanya saja tipe datanya diubah dari **int** menjadi **double**.

Class ini memiliki atribut **data** dan **next**, serta dilengkapi constructor, getter, dan setter.

B. Tambahan Code Di File Struktur List

```
//tugas
// Tambahan struktur list untuk tipe double
class StrukturListDouble {
    NodeDouble HEAD;

    public boolean isEmpty() {
        return (HEAD == null);
    }

    public void display() {
        NodeDouble curNode = HEAD;
        while (curNode != null) {
            System.out.print(curNode.getData() + " ");
            curNode = curNode.getNext();
        }
        System.out.println();
    }

    public void addHead(double data) {
        NodeDouble newNode = new NodeDouble(data);
        if (isEmpty()) {
            HEAD = newNode;
        } else {
            newNode.setNext(HEAD);
            HEAD = newNode;
        }
    }

    public void addTail(double data) {
        NodeDouble newNode = new NodeDouble(data);
        if (isEmpty()) {
            HEAD = newNode;
        } else {
            NodeDouble curNode = HEAD;
            while (curNode.getNext() != null) {
                curNode = curNode.getNext();
            }
            curNode.setNext(newNode);
        }
    }

    public void addMid(double data, int position) {
        NodeDouble newNode = new NodeDouble(data);
        if (HEAD == null || position == 1) {
            newNode.setNext(HEAD);
            HEAD = newNode;
        } else {
            NodeDouble curNode = HEAD;
            int i = 1;
            while (curNode != null && i < position - 1) {
                curNode = curNode.getNext();
                i++;
            }
            if (curNode != null) {
                newNode.setNext(curNode.getNext());
                curNode.setNext(newNode);
            }
        }
    }
}
```

Penjelasan:

2. Class **StrukturListDouble**

Class **StrukturListDouble** merupakan struktur linked list yang menangani data bertipe **double**.

Di dalam class ini terdapat beberapa method penting:

❖ **addHead(double data)**

Menambahkan node baru di depan list. Node baru akan menjadi **HEAD**, dan menunjuk ke node sebelumnya.

❖ **addTail(double data)**

Menambahkan node baru di akhir list. Pointer akan bergerak dari **HEAD** hingga ke node terakhir, lalu menyisipkan node baru di belakangnya.

❖ **addMid(double data, int position)**

Menyisipkan node baru di tengah, berdasarkan posisi tertentu. Jika posisi = 1, maka node langsung menjadi **HEAD**.

❖ **display()**

Menampilkan seluruh isi list dari **HEAD** sampai node terakhir.

C. Code StrukturListTugas

```
TugasDanPertemuanCoding > TugasPertemuan4 > J StrukturListTugas.java > ...
1  public class StrukturListTugas {
    Run | Debug
2  public static void main(String[] args) {
3      StrukturListDouble listA = new StrukturListDouble();
4      listA.addTail(data:2.1);
5      listA.addTail(data:3.4);
6      listA.addTail(data:4.5);
7      System.out.println("Deret A:");
8      listA.display();
9
10     StrukturListDouble listB = new StrukturListDouble();
11     listB.addHead(data:5.5);
12     listB.addHead(data:4.5);
13     listB.addHead(data:1.1);
14     listB.addHead(data:2.1);
15     listB.addHead(data:3.4);
16     System.out.println("\nDeret B:");
17     listB.display();
18 }
19 }
20 }
```

Penjelasan:

Class **StrukturListTugas** adalah bagian utama dari program yang digunakan untuk menguji fitur-fitur pada **StrukturListDouble**.

1. Deret A

- ❖ **StrukturListDouble listA = new StrukturListDouble();**
- ❖ **listA.addTail(2.1);**
- ❖ **listA.addTail(3.4);**
- ❖ **listA.addTail(4.5);**
- ❖ **listA.display();**

Kode di atas membuat list baru dan menambahkan tiga angka pecahan di bagian akhir list menggunakan **addTail()**. Hasil akhir list A: **2.1, 3.4, 4.5**

2. Deret B

- ❖ **StrukturListDouble listB = new StrukturListDouble();**
- ❖ **listB.addHead(5.5);**
- ❖ **listB.addHead(4.5);**
- ❖ **listB.addHead(1.1);**
- ❖ **listB.addHead(2.1);**
- ❖ **listB.addHead(3.4);**
- ❖ **listB.display();**

Pada deret B, data dimasukkan dari depan menggunakan **addHead()**, sehingga urutannya terbalik dari input.

Hasil akhir list B: **3.4, 2.1, 1.1, 4.5, 5.5** Method **display()** digunakan untuk menampilkan isi list ke layar.

Hasil:

```
Deret A:
2.1 3.4 4.5

Deret B:
3.4 2.1 1.1 4.5 5.5
PS D:\KuliahSMT4\Praktikum Pemrograman 1>
```