Laporan Praktikum

Struktur Data

Disusun Oleh :

SASKIA ALIFAH (2411531002)

Dosen Pengampu : Dr. Wahyudi, S.T, M.T

Departemen Informatika

Fakultas Teknologi Informasi

Universitas Andalas

Tahun 2025

SINGLE LINKED LIST

A. Tujuan Praktikum

1. Memahami konsep dasar Single Linked List (SLL) sebagai salah satu struktur data dinamis.
2. Mengimplementasikan operasi-operasi dasar pada SLL seperti penambahan, penghapusan, dan pencarian node menggunakan bahasa pemrograman Java.
3. Menganalisis cara kerja dan efek dari setiap operasi SLL melalui pengamatan hasil keluaran program.
4. Melatih kemampuan logika dan pemrograman dalam mengelola struktur data berbasis pointer secara efisien serta aman.

B. Pendahuluan

Single Linked List (SLL) merupakan salah satu struktur data linear yang terdiri dari rangkaian node, di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya. Struktur ini sangat penting dalam dunia pemrograman karena mampu mengelola data secara dinamis, yang berarti ukurannya dapat bertambah atau berkurang sesuai kebutuhan aplikasi tanpa harus mengalokasikan ulang seluruh memori seperti pada array. Dengan demikian, SLL sangat efisien untuk aplikasi yang membutuhkan fleksibilitas dalam pengelolaan data.

Pada SLL, setiap node terdiri dari dua bagian utama: data dan pointer ke node berikutnya (next). Traversal atau penelusuran hanya bisa dilakukan satu arah, yaitu dari head ke tail, karena tidak ada pointer ke node sebelumnya. Hal ini membuat operasi penambahan dan penghapusan di depan list menjadi sangat cepat, namun pencarian atau penghapusan di posisi tertentu membutuhkan traversal satu per satu dari awal list.

SLL banyak digunakan dalam berbagai aplikasi nyata, seperti pada implementasi antrian (queue), tumpukan (stack), pengelolaan memori dinamis, serta pada sistem operasi dan aplikasi basis data. Dengan memahami SLL, seorang programmer dapat membangun struktur data yang fleksibel dan mudah dimodifikasi sesuai kebutuhan aplikasi.

Salah satu keunggulan utama SLL adalah kemudahan dalam menambah atau menghapus elemen di bagian depan list. Namun, kelemahannya adalah tidak efisien untuk akses acak atau pencarian data di posisi tertentu karena harus dilakukan traversal dari awal. Oleh karena itu, pemilihan SLL sebagai struktur data sangat tergantung pada kebutuhan aplikasi yang akan dibangun.

Dalam praktikum ini, operasi-operasi dasar pada SLL diimplementasikan menggunakan bahasa pemrograman Java. Beberapa operasi yang dipelajari antara lain penambahan node di depan, belakang, dan posisi tertentu; penghapusan node pada head,

tail, dan posisi tertentu; serta pencarian data dalam list. Setiap operasi dipecah ke dalam kelas-kelas terpisah agar lebih terstruktur dan mudah dipahami.

Kode program yang digunakan terdiri dari beberapa kelas utama, yaitu NodeSLL sebagai representasi node, TambahSLL untuk operasi penambahan node, HapusSLL untuk operasi penghapusan node, dan PencarianSLL untuk operasi pencarian data. Setiap kelas memiliki fungsi-fungsi spesifik yang saling melengkapi dalam pengelolaan SLL.

Pemahaman mendalam terhadap setiap operasi sangat penting agar dapat menghindari kesalahan seperti kehilangan referensi node, memory leak, atau infinite loop akibat pointer yang tidak diatur dengan benar. Praktikum ini juga menekankan pentingnya pengujian setiap fungsi dengan berbagai skenario input untuk memastikan keandalan dan stabilitas program.

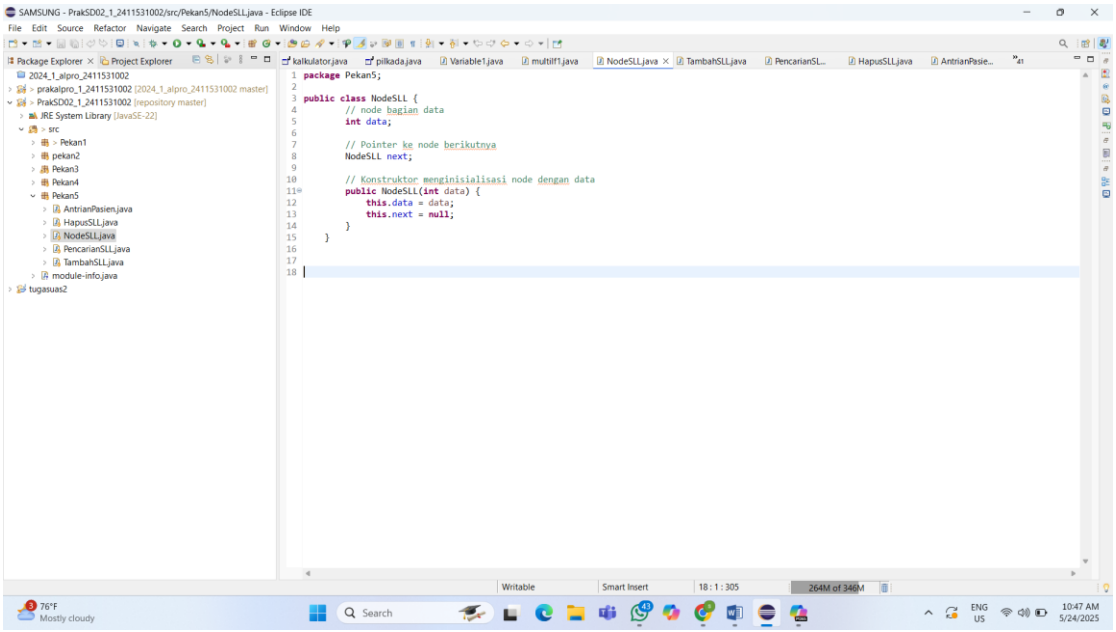
Selain aspek teknis, penguasaan SLL juga sangat penting untuk memahami struktur data yang lebih kompleks, seperti double linked list, circular linked list, hingga struktur data berbasis pohon dan graf. Dengan menguasai konsep dan implementasi SLL, mahasiswa akan lebih siap menghadapi tantangan pengelolaan data yang lebih besar dan kompleks di dunia nyata.

C. Metode Praktikum

1. Kelas NodeSLL

SYNTAX :

```
package Pekan5;

public class NodeSLL {
        // node bagian data
        int data;

        // Pointer ke node berikutnya
        NodeSLL next;

        // Konstruktor menginisialisasi node dengan data
        public NodeSLL(int data) {
            this.data = data;
            this.next = null;
        }
    }
```



Deskripsi Kelas

Kelas NodeSLL adalah kelas dasar yang merepresentasikan satu simpul (node) dalam Single Linked List. Kelas ini terdiri dari dua atribut utama, yaitu:

data: Menyimpan nilai/data yang akan dimasukkan ke dalam node (dalam praktikum ini bertipe integer).

next: Merupakan pointer (referensi) yang menunjuk ke node berikutnya dalam list. Jika node ini adalah node terakhir, maka next akan bernilai null.

kelas ini menerima satu parameter (data) dan langsung menginisialisasi atribut data serta mengatur next menjadi null. Dengan demikian, setiap kali objek NodeSLL dibuat, node tersebut siap untuk dihubungkan ke node lain dalam list.

Kelas ini sangat penting karena menjadi dasar dari seluruh operasi pada SLL. Semua operasi penambahan, penghapusan, dan pencarian node akan bekerja dengan memanipulasi objek-objek dari kelas ini. Dengan adanya pointer next, setiap node dapat saling terhubung membentuk rantai node yang dapat tumbuh dan menyusut sesuai kebutuhan.

Fungsi Kelas

Kelas ini tidak memiliki fungsi lain selain konstruktor, namun merupakan pondasi dari semua operasi pada SLL. Semua penambahan, penghapusan, dan pencarian node akan bekerja dengan memanipulasi objek-objek dari kelas ini.

Output

Kelas NodeSLL sendiri tidak menghasilkan output secara langsung. Namun, tanpa kelas ini, tidak ada struktur dasar yang bisa digunakan untuk menyimpan dan menghubungkan data dalam SLL.

2. Kelas TambahSLL

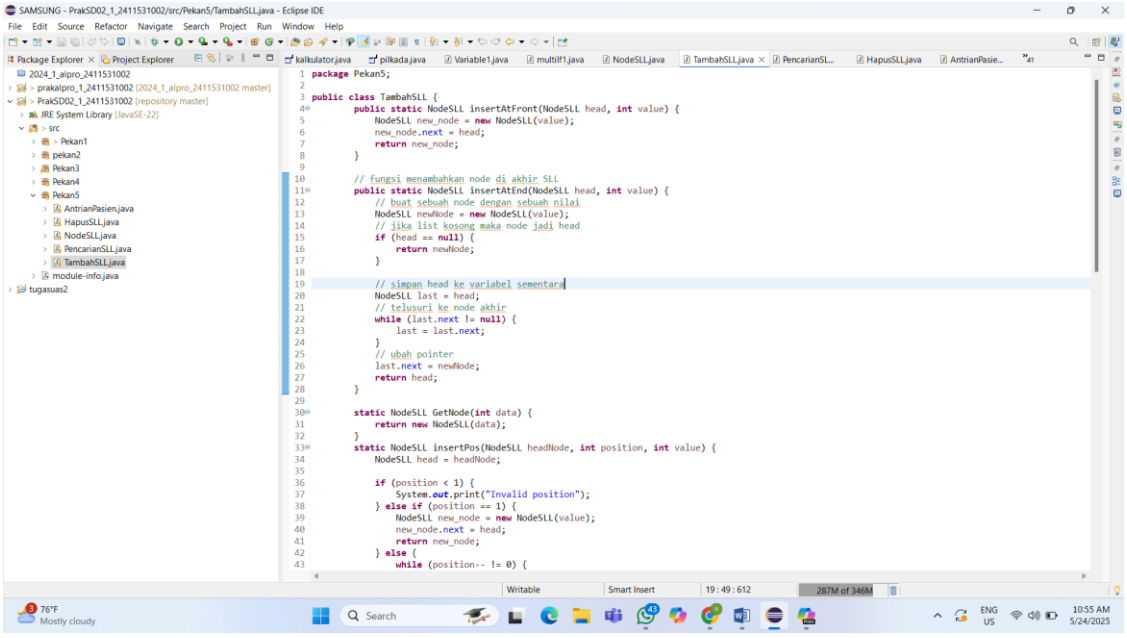
SYNTAX :

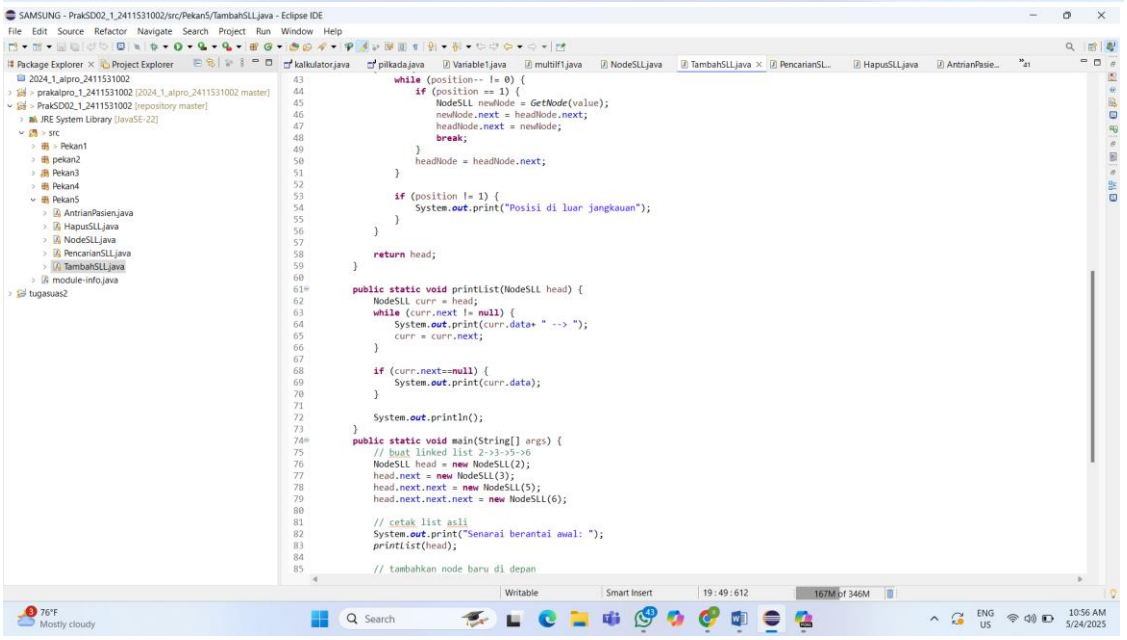
```
package Pekan5;

public class TambahSLL {
        public static NodeSLL insertAtFront(NodeSLL head, int value) {
            NodeSLL new_node = new NodeSLL(value);
            new_node.next = head;
            return new_node;
        }

        // fungsi menambahkan node di akhir SLL
        public static NodeSLL insertAtEnd(NodeSLL head, int value) {
            // buat sebuah node dengan sebuah nilai
            NodeSLL newNode = new NodeSLL(value);
            // jika list kosong maka node jadi head
            if (head == null) {
                return newNode;
```

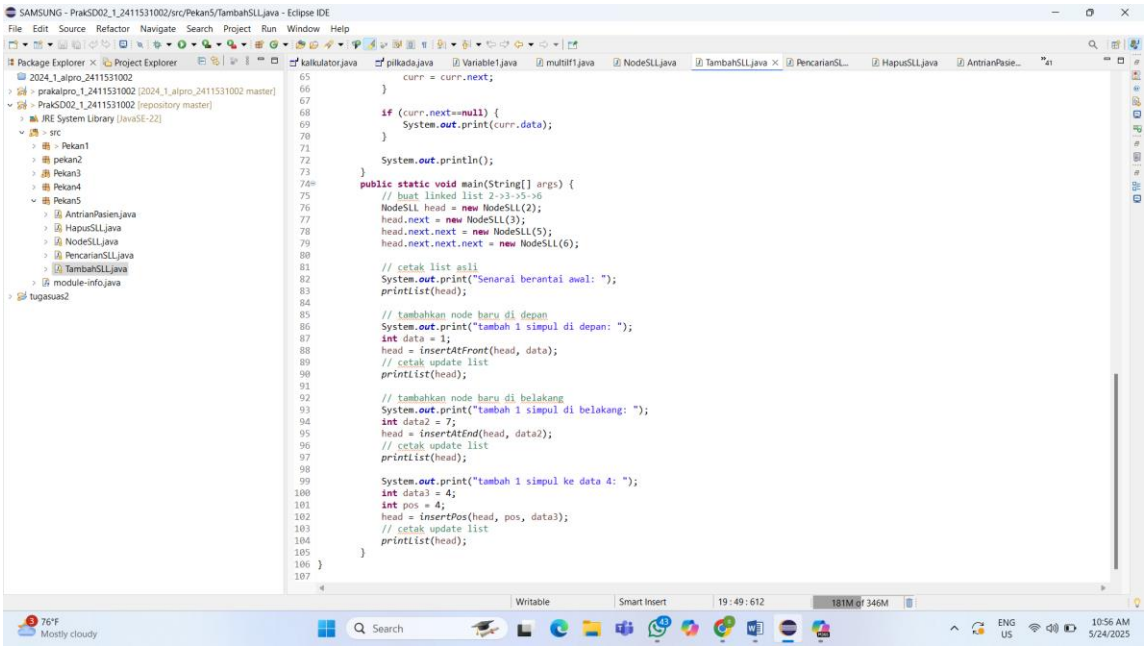
```
    }

    // simpan head ke variabel sementara
    NodeSLL last = head;
    // telusuri ke node akhir
    while (last.next != null) {
        last = last.next;
    }
    // ubah pointer
    last.next = newNode;
    return head;
}

static NodeSLL GetNode(int data) {
    return new NodeSLL(data);
}
static NodeSLL insertPos(NodeSLL headNode, int position, int value) {
    NodeSLL head = headNode;

    if (position < 1) {
        System.out.print("Invalid position");
    } else if (position == 1) {
        NodeSLL new_node = new NodeSLL(value);
        new_node.next = head;
        return new_node;
    } else {
        while (position-- != 0) {
            if (position == 1) {
                NodeSLL newNode = GetNode(value);
                newNode.next = headNode.next;
                headNode.next = newNode;
                break;
            }
            headNode = headNode.next;
        }

        if (position != 1) {
            System.out.print("Posisi di luar jangkauan");
        }
    }

    return head;
}
```

```
public static void printList(NodeSLL head) {
    NodeSLL curr = head;
    while (curr.next != null) {
        System.out.print(curr.data+ " --> ");
        curr = curr.next;
    }

    if (curr.next==null) {
        System.out.print(curr.data);
    }

    System.out.println();
}
public static void main(String[] args) {
    // buat linked list 2->3->5->6
    NodeSLL head = new NodeSLL(2);
    head.next = new NodeSLL(3);
    head.next.next = new NodeSLL(5);
    head.next.next.next = new NodeSLL(6);

    // cetak list asli
    System.out.print("Senarai berantai awal: ");
    printList(head);

    // tambahkan node baru di depan
    System.out.print("tambah 1 simpul di depan: ");
    int data = 1;
    head = insertAtFront(head, data);
    // cetak update list
    printList(head);

    // tambahkan node baru di belakang
    System.out.print("tambah 1 simpul di belakang: ");
    int data2 = 7;
    head = insertAtEnd(head, data2);
    // cetak update list
    printList(head);

    System.out.print("tambah 1 simpul ke data 4: ");
    int data3 = 4;
    int pos = 4;
    head = insertPos(head, pos, data3);
    // cetak update list
    printList(head);
```

```
        }
}
```


SAMSUNG - PrakSD02_1_2411531002/src/Pekan5/TambahSLL.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
2024_1_alpro_2411531002
prakalpro_1_2411531002 [2024_1_alpro_2411531002 master]
PrakSD02_1_2411531002 [repository master]
JRE System Library [JavaSE-22]
src
Pekan1
pekan2
Pekan3
Pekan4
Pekan5
AntrianPasien.java
HapusSLL.java
NodeSLL.java
PencarianSLL.java
TambahSLL.java
module-info.java
tugasuas2
kalkulator.java
pilkada.java
Variable1.java
multilf1.java
NodeSLL.java
TambahSLL.java
PencarianSL...
HapusSLL.java
AntrianPasie...
package Pekan5;
public class TambahSLL {
public static NodeSLL insertAtFront(NodeSLL head, int value) {
NodeSLL new_node = new NodeSLL(value);
new_node.next = head;
return new_node;
}
// fungsi menambahkan node di akhir SLL
public static NodeSLL insertAtEnd(NodeSLL head, int value) {
// buat sebuah node dengan sebuah nilai
NodeSLL newNode = new NodeSLL(value);
// jika list kosong maka node jadi head
if (head == null) {
return newNode;
}
// simpan head ke variabel sementara
NodeSLL last = head;
// telusuri ke node akhir
while (last.next != null) {
last = last.next;
}
// ubah pointer
last.next = newNode;
return head;
}
static NodeSLL GetNode(int data) {
return new NodeSLL(data);
}
static NodeSLL insertPos(NodeSLL headNode, int position, int value) {
NodeSLL head = headNode;
if (position < 1) {
System.out.print("Invalid position");
} else if (position == 1) {
NodeSLL new_node = new NodeSLL(value);
new_node.next = head;
return new_node;
} else {
while (position-- != 0) {
Writable
Smart Insert
19 : 49 : 612
287M of 346M
76°F
Mostly cloudy
Search
ENG
US
10:55 AM
5/24/2025


SAMSUNG - PrakSD02_1_2411531002/src/Pekan5/TambahSLL.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
2024_1_alpro_2411531002
prakalpro_1_2411531002 [2024_1_alpro_2411531002 master]
PrakSD02_1_2411531002 [repository master]
JRE System Library [JavaSE-22]
src
Pekan1
pekan2
Pekan3
Pekan4
Pekan5
AntrianPasien.java
HapusSLL.java
NodeSLL.java
PencarianSLL.java
TambahSLL.java
module-info.java
tugasuas2
kalkulator.java
pilkada.java
Variable1.java
multilf1.java
NodeSLL.java
TambahSLL.java
PencarianSL...
HapusSLL.java
AntrianPasie...
while (position-- != 0) {
if (position == 1) {
NodeSLL newNode = GetNode(value);
newNode.next = headNode.next;
headNode.next = newNode;
break;
}
headNode = headNode.next;
}
if (position != 1) {
System.out.print("Posisi di luar jangkauan");
}
}
return head;
}
public static void printList(NodeSLL head) {
NodeSLL curr = head;
while (curr.next != null) {
System.out.print(curr.data+ " --> ");
curr = curr.next;
}
if (curr.next==null) {
System.out.print(curr.data);
}
System.out.println();
}
public static void main(String[] args) {
// buat linked list 2->3->5->6
NodeSLL head = new NodeSLL(2);
head.next = new NodeSLL(3);
head.next.next = new NodeSLL(5);
head.next.next.next = new NodeSLL(6);
// cetak list asli
System.out.print("Senarai berantai awal: ");
printList(head);
// tambahkan node baru di depan
Writable
Smart Insert
19 : 49 : 612
167M of 346M
76°F
Mostly cloudy
Search
ENG
US
10:56 AM
5/24/2025



Deskripsi Kelas

Kelas TambahSLL berisi kumpulan fungsi statis untuk menambah node pada Single Linked List. Penambahan node bisa dilakukan di depan (head), di belakang (tail), maupun pada posisi tertentu di dalam list. Kelas ini juga menyediakan fungsi untuk mencetak isi list.

Penjelasan Fungsi

Kelas TambahSLL menyediakan berbagai metode untuk menambah node ke dalam Single Linked List, baik di depan, di belakang, maupun di posisi tertentu. Berikut penjelasan detail setiap fungsi:

- **insertAtFront(NodeSLL head, int value)**
  Fungsi ini menambah node baru di bagian depan (head) list. Node baru dibuat, kemudian pointer next dari node tersebut diarahkan ke head lama, dan node baru ini menjadi head yang baru. Operasi ini sangat efisien karena hanya membutuhkan waktu konstan tanpa traversal.
  Kasus khusus: Jika list kosong, node baru langsung menjadi head.
- **insertAtEnd(NodeSLL head, int value)**
  Fungsi ini menambah node di bagian akhir list. Jika list kosong, node baru langsung menjadi head. Jika tidak, fungsi akan menelusuri seluruh node hingga menemukan node terakhir, lalu menambahkan node baru di ujung dengan mengatur pointer next node terakhir ke node baru. Proses ini membutuhkan waktu linear karena traversal dari head hingga tail.
  Kasus khusus: Jika list hanya satu node, node baru langsung menjadi next dari node head.

- **insertPos(NodeSLL headNode, int position, int value)**
  Fungsi ini menambah node pada posisi tertentu dalam list. Jika posisi adalah 1, node baru langsung menjadi head. Jika posisi lebih kecil dari 1, akan muncul pesan error "Invalid position". Jika posisi melebihi panjang list, akan muncul pesan "Posisi di luar jangkauan". Fungsi ini melakukan traversal hingga posisi yang diinginkan, lalu menyisipkan node baru di posisi tersebut.
  Kasus khusus: Jika posisi yang dimasukkan tidak valid atau melebihi panjang list, node tidak akan ditambahkan.
- **printList(NodeSLL head)**
  Fungsi ini menelusuri seluruh node dari head hingga tail dan mencetak data setiap node secara berurutan, sehingga memudahkan pengguna untuk melihat isi list setelah setiap operasi.

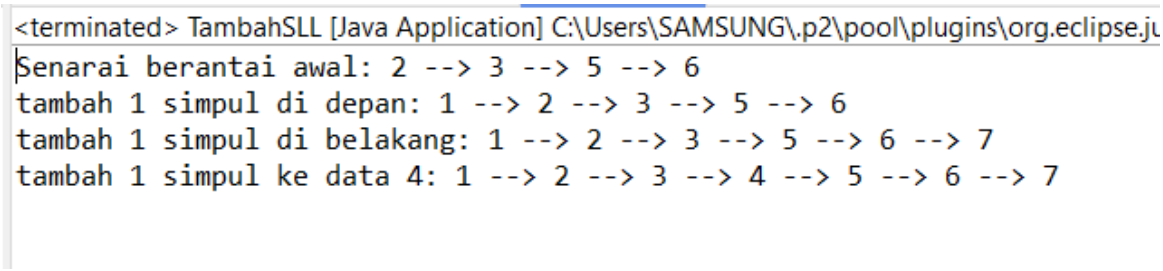
Penjelasan Logika Traversal dan Error Handling

Pada setiap operasi penambahan, pointer akan bergerak dari head menuju node berikutnya hingga mencapai posisi yang diinginkan. Jika posisi yang dimasukkan tidak valid (misal lebih kecil dari 1 atau melebihi panjang list), maka akan muncul pesan error. Hal ini penting untuk mencegah terjadinya error atau kerusakan pada struktur list.

Penjelasan Fungsi Main

Pada fungsi main, dibuatlah linked list awal dengan data 2 → 3 → 5 → 6. Kemudian, dilakukan penambahan node di depan (data 1), di belakang (data 7), dan di posisi ke-4 (data 4). Setelah setiap operasi, list dicetak menggunakan printList.

Output :



Analisis Output:

Setiap kali fungsi penambahan dijalankan, node baru akan muncul di posisi yang diinginkan, dan hasilnya langsung terlihat pada output yang dicetak ke layar. Jika terjadi error pada posisi penambahan, output akan menampilkan pesan error.

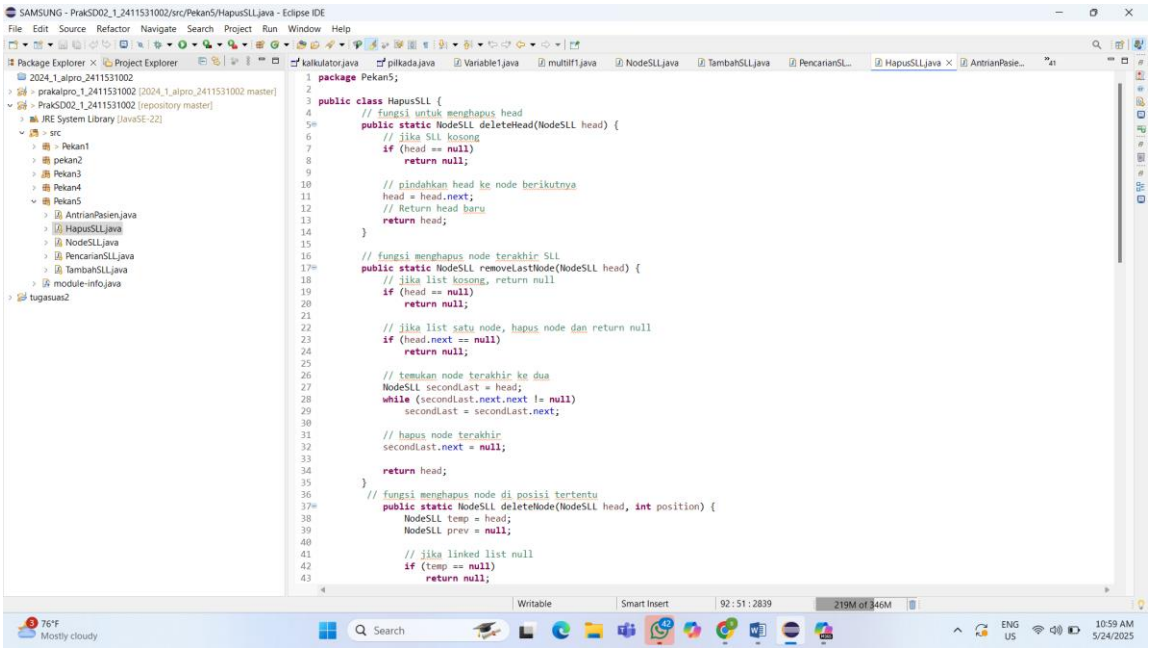
3. Kelas HapusSLL

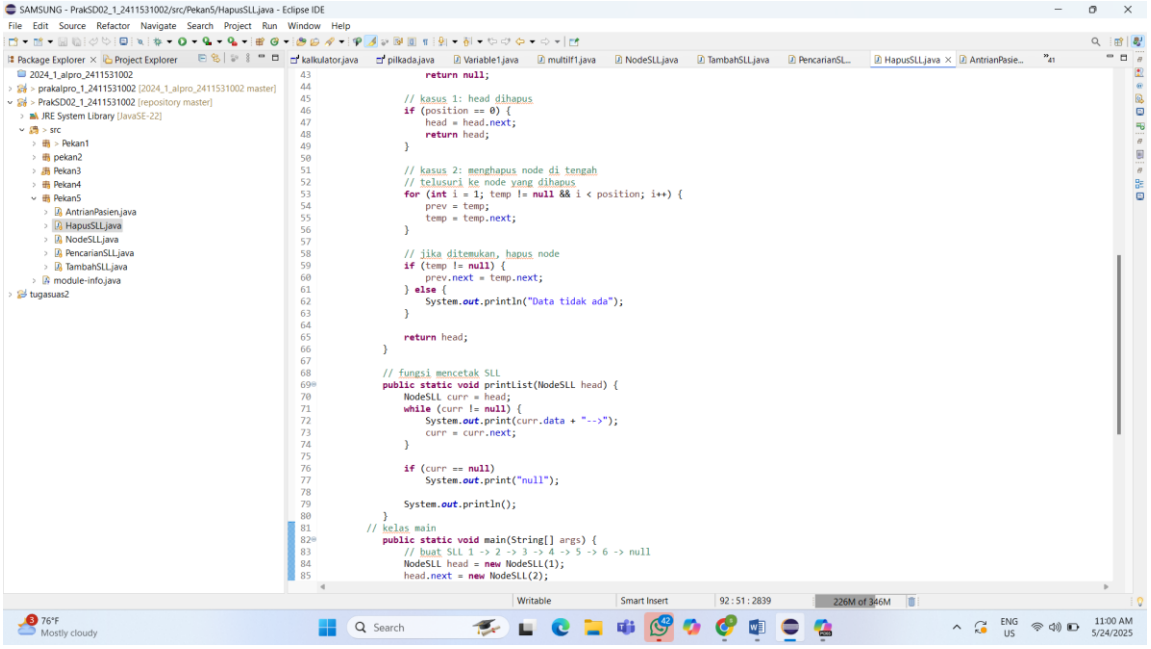
SYNTAX :

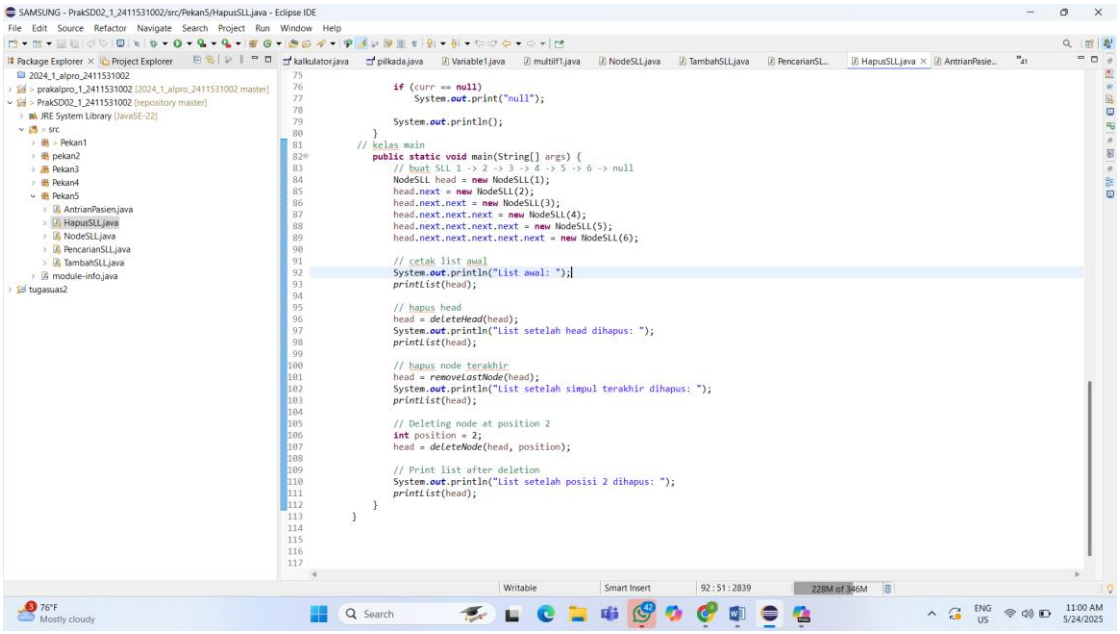
```
package Pekan5;

public class HapusSLL {
        // fungsi untuk menghapus head
        public static NodeSLL deleteHead(NodeSLL head) {
            // jika SLL kosong
            if (head == null)
                return null;

            // pindahkan head ke node berikutnya
            head = head.next;
            // Return head baru
            return head;
        }

        // fungsi menghapus node terakhir SLL
        public static NodeSLL removeLastNode(NodeSLL head) {
            // jika list kosong, return null
            if (head == null)
                return null;

            // jika list satu node, hapus node dan return null
            if (head.next == null)
                return null;

            // temukan node terakhir ke dua
            NodeSLL secondLast = head;
            while (secondLast.next.next != null)
                secondLast = secondLast.next;

            // hapus node terakhir
            secondLast.next = null;

            return head;
        }
         // fungsi menghapus node di posisi tertentu
            public static NodeSLL deleteNode(NodeSLL head, int position) {
                NodeSLL temp = head;
                NodeSLL prev = null;

                // jika linked list null
                if (temp == null)
                    return null;
```

```
        // kasus 1: head dihapus
        if (position == 0) {
            head = head.next;
            return head;
        }

        // kasus 2: menghapus node di tengah
        // telusuri ke node yang dihapus
        for (int i = 1; temp != null && i < position; i++) {
            prev = temp;
            temp = temp.next;
        }

        // jika ditemukan, hapus node
        if (temp != null) {
            prev.next = temp.next;
        } else {
            System.out.println("Data tidak ada");
        }

        return head;
    }

    // fungsi mencetak SLL
    public static void printList(NodeSLL head) {
        NodeSLL curr = head;
        while (curr != null) {
            System.out.print(curr.data + "-->");
            curr = curr.next;
        }

        if (curr == null)
            System.out.print("null");

        System.out.println();
    }
// kelas main
    public static void main(String[] args) {
        // buat SLL 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
        NodeSLL head = new NodeSLL(1);
        head.next = new NodeSLL(2);
        head.next.next = new NodeSLL(3);
        head.next.next.next = new NodeSLL(4);
```

```
        head.next.next.next.next = new NodeSLL(5);
        head.next.next.next.next.next = new NodeSLL(6);

        // cetak list awal
        System.out.println("List awal: ");
        printList(head);

        // hapus head
        head = deleteHead(head);
        System.out.println("List setelah head dihapus: ");
        printList(head);

        // hapus node terakhir
        head = removeLastNode(head);
        System.out.println("List setelah simpul terakhir dihapus: ");
        printList(head);

        // Deleting node at position 2
        int position = 2;
        head = deleteNode(head, position);

        // Print list after deletion
        System.out.println("List setelah posisi 2 dihapus: ");
        printList(head);
    }
}
```


SAMSUNG - PrakSD02_1_2411531002/src/Pekan5/HapusSLL.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
2024_1_alpro_2411531002
prakalpro_1_2411531002 [2024_1_alpro_2411531002 master]
PrakSD02_1_2411531002 [repository master]
JRE System Library [JavaSE-22]
src
Pekan1
pekan2
Pekan3
Pekan4
Pekan5
AntrianPasien.java
HapusSLL.java
NodeSLL.java
PencarianSLL.java
TambahSLL.java
module-info.java
tugasuas2
kalkulator.java
pilkada.java
Variable1.java
multilf1.java
NodeSLL.java
TambahSLL.java
PencarianSL...
HapusSLL.java
AntrianPasie...
1 package Pekan5;
2
3 public class HapusSLL {
4 // fungsi untuk menghapus head
5 public static NodeSLL deleteHead(NodeSLL head) {
6 // jika SLL kosong
7 if (head == null)
8 return null;
9
10 // pindahkan head ke node berikutnya
11 head = head.next;
12 // Return head baru
13 return head;
14 }
15
16 // fungsi menghapus node terakhir SLL
17 public static NodeSLL removeLastNode(NodeSLL head) {
18 // jika list kosong, return null
19 if (head == null)
20 return null;
21
22 // jika list satu node, hapus node dan return null
23 if (head.next == null)
24 return null;
25
26 // temukan node terakhir ke dua
27 NodeSLL secondLast = head;
28 while (secondLast.next.next != null)
29 secondLast = secondLast.next;
30
31 // hapus node terakhir
32 secondLast.next = null;
33
34 return head;
35 }
36 // fungsi menghapus node di posisi tertentu
37 public static NodeSLL deleteNode(NodeSLL head, int position) {
38 NodeSLL temp = head;
39 NodeSLL prev = null;
40
41 // jika linked list null
42 if (temp == null)
43 return null;
Writable
Smart Insert
92 : 51 : 2839
219M of 346M
76°F
Mostly cloudy
Search
ENG
US
10:59 AM
5/24/2025


SAMSUNG - PrakSD02_1_2411531002/src/Pekan5/HapusSLL.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Project Explorer
2024_1_alpro_2411531002
prakalpro_1_2411531002 [2024_1_alpro_2411531002 master]
PrakSD02_1_2411531002 [repository master]
JRE System Library [JavaSE-22]
src
Pekan1
pekan2
Pekan3
Pekan4
Pekan5
AntrianPasien.java
HapusSLL.java
NodeSLL.java
PencarianSLL.java
TambahSLL.java
module-info.java
tugasuas2
kalkulator.java
pilkada.java
Variable1.java
multilf1.java
NodeSLL.java
TambahSLL.java
PencarianSL...
HapusSLL.java
AntrianPasie...
return null;
// kasus 1: head dihapus
if (position == 0) {
head = head.next;
return head;
}
// kasus 2: menghapus node di tengah
// telusuri ke node yang dihapus
for (int i = 1; temp != null && i < position; i++) {
prev = temp;
temp = temp.next;
}
// jika ditemukan, hapus node
if (temp != null) {
prev.next = temp.next;
} else {
System.out.println("Data tidak ada");
}
return head;
}
// fungsi mencetak SLL
public static void printList(NodeSLL head) {
NodeSLL curr = head;
while (curr != null) {
System.out.print(curr.data + "-->");
curr = curr.next;
}
if (curr == null)
System.out.print("null");
System.out.println();
}
// kelas main
public static void main(String[] args) {
// buat SLL 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
NodeSLL head = new NodeSLL(1);
head.next = new NodeSLL(2);
Writable
Smart Insert
92 : 51 : 2839
226M of 346M
76°F
Mostly cloudy
Search
ENG
US
11:00 AM
5/24/2025



Deskripsi Kelas

Kelas HapusSLL berisi fungsi-fungsi untuk menghapus node dari Single Linked List. Penghapusan dapat dilakukan pada head, tail, atau posisi tertentu dalam list. Kelas ini juga menyediakan fungsi untuk mencetak isi list.

Penjelasan Fungsi

Kelas HapusSLL berfungsi untuk menghapus node dari Single Linked List dengan menyediakan beberapa metode utama:

- **deleteHead(NodeSLL head)**
  Fungsi ini menghapus node paling depan (head) dari list. Jika list kosong, fungsi mengembalikan null. Jika tidak, pointer head dipindahkan ke node berikutnya, sehingga node lama otomatis terputus dan akan dihapus oleh garbage collector Java.
  Kasus khusus: Jika hanya ada satu node, setelah dihapus list akan menjadi kosong.
- **removeLastNode(NodeSLL head)**
  Fungsi ini menghapus node paling belakang (tail) dari list. Jika list kosong atau hanya berisi satu node, fungsi mengembalikan null. Jika tidak, fungsi melakukan traversal untuk menemukan node sebelum terakhir (secondLast), lalu mengatur pointer next node tersebut menjadi null, sehingga node terakhir terputus dari list.
  Kasus khusus: Jika hanya ada satu node, setelah dihapus list akan menjadi kosong.
- **deleteNode(NodeSLL head, int position)**
  Fungsi ini menghapus node pada posisi tertentu. Jika posisi adalah 0, head dihapus. Untuk posisi lain, fungsi melakukan traversal hingga posisi yang diinginkan, lalu mengatur pointer node sebelumnya agar melewati node yang akan

dihapus. Jika posisi tidak ditemukan, akan muncul pesan "Data tidak ada".
Kasus khusus: Jika posisi yang dimasukkan tidak valid atau melebihi panjang list, node tidak akan dihapus dan akan muncul pesan error.

- **printList(NodeSLL head)**
  Fungsi ini menampilkan seluruh isi list dari head hingga tail, memudahkan pengguna untuk memverifikasi hasil operasi penghapusan.

Penjelasan Logika Traversal dan Error Handling

Pada setiap operasi penghapusan, pointer berpindah dari head menuju node sebelum node yang akan dihapus. Jika posisi yang dimasukkan tidak valid atau list kosong, maka fungsi akan mengembalikan null dan menampilkan pesan error. Hal ini penting untuk menjaga integritas struktur list dan mencegah terjadinya error.

Penjelasan Fungsi Main

Pada fungsi main, dibuat linked list awal 1 → 2 → 3 → 4 → 5 → 6. Kemudian dilakukan penghapusan head, penghapusan node terakhir, dan penghapusan node pada posisi ke-2. Setelah setiap operasi, hasil list dicetak menggunakan printList.

Output :

```
<terminated> HapusSLL [Java Application] C:\Users\SAMSUNG\.p2\pool\p
List awal:
1-->2-->3-->4-->5-->6-->null
List setelah head dihapus:
2-->3-->4-->5-->6-->null
List setelah simpul terakhir dihapus:
2-->3-->4-->5-->null
List setelah posisi 2 dihapus:
2-->4-->5-->null
```

Analisis Output:

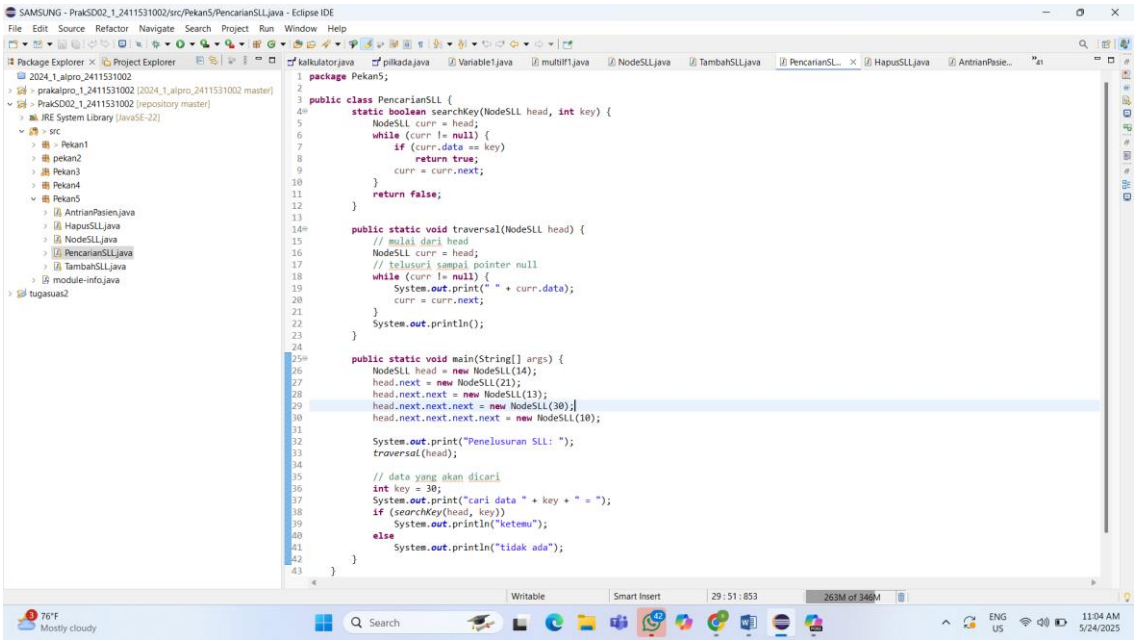
Setiap kali fungsi penghapusan dijalankan, node yang dihapus akan hilang dari list, dan hasilnya langsung terlihat pada output yang dicetak ke layar. Jika posisi penghapusan tidak valid, output akan menampilkan pesan error.

4. **Kelas PencarianSLL**

SYNTAX :

```
package Pekan5;
```

```
public class PencarianSLL {
        static boolean searchKey(NodeSLL head, int key) {
            NodeSLL curr = head;
            while (curr != null) {
                if (curr.data == key)
                    return true;
                curr = curr.next;
            }
            return false;
        }

        public static void traversal(NodeSLL head) {
            // mulai dari head
            NodeSLL curr = head;
            // telusuri sampai pointer null
            while (curr != null) {
                System.out.print(" " + curr.data);
                curr = curr.next;
            }
            System.out.println();
        }

        public static void main(String[] args) {
            NodeSLL head = new NodeSLL(14);
            head.next = new NodeSLL(21);
            head.next.next = new NodeSLL(13);
            head.next.next.next = new NodeSLL(30);
            head.next.next.next.next = new NodeSLL(10);

            System.out.print("Penelusuran SLL: ");
            traversal(head);

            // data yang akan dicari
            int key = 30;
            System.out.print("cari data " + key + " = ");
            if (searchKey(head, key))
                System.out.println("ketemu");
            else
                System.out.println("tidak ada");
        }
    }
```



Deskripsi Kelas

Kelas PencarianSLL berisi fungsi pencarian data dalam Single Linked List dan fungsi untuk menelusuri isi list. Kelas ini berguna untuk mengetahui apakah sebuah data terdapat dalam list dan untuk menampilkan seluruh isi list.

Penjelasan Fungsi

Kelas PencarianSLL menyediakan metode untuk melakukan pencarian data dalam Single Linked List:

- **searchKey(NodeSLL head, int key)**
  Fungsi ini melakukan pencarian data dengan nilai tertentu (key) dalam list. Fungsi melakukan traversal dari head hingga tail, membandingkan setiap data node dengan key. Jika ditemukan, fungsi mengembalikan true, jika tidak ditemukan hingga akhir list, mengembalikan false.
  Kasus khusus: Jika list kosong, fungsi langsung mengembalikan false.
- **traversal(NodeSLL head)**
  Fungsi ini menampilkan seluruh isi list dari head hingga tail, sehingga pengguna dapat melihat isi list sebelum atau sesudah operasi pencarian.

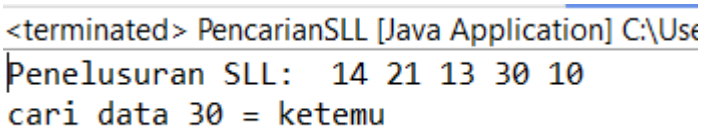
Penjelasan Logika Traversal dan Error Handling

Proses pencarian dilakukan dengan menelusuri setiap node dari head hingga tail. Jika data yang dicari tidak ditemukan, fungsi akan mengembalikan false, menandakan bahwa data tersebut tidak ada di dalam list.

Penjelasan Fungsi Main

Pada fungsi main, dibuat linked list dengan data 14 → 21 → 13 → 30 → 10. Fungsi traversal digunakan untuk menampilkan isi list, kemudian dilakukan pencarian data 30 menggunakan searchKey. Hasil pencarian dicetak ke layar.

Output :



Analisis Output :

Jika data yang dicari ada dalam list, maka output akan menampilkan "ketemu". Jika tidak, akan menampilkan "tidak ada".

D. Kesimpulan Praktikum

Praktikum Single Linked List ini membuktikan bahwa SLL adalah struktur data yang sangat fleksibel untuk pengelolaan data secara dinamis. Dengan memahami dan mengimplementasikan berbagai operasi dasar seperti penambahan, penghapusan, dan pencarian node, mahasiswa dapat memahami cara kerja pointer dan traversal dalam list. Setiap kelas dalam program memiliki peran penting dalam membangun dan memanipulasi SLL, serta memberikan pengalaman praktis dalam pengelolaan struktur data dinamis. Praktikum ini memperkuat pemahaman konsep linked list dan meningkatkan kemampuan pemrograman serta problem solving, yang sangat berguna dalam pengembangan aplikasi yang lebih kompleks di masa mendatang.