LAPORAN PRAKTIKUM
Modul 4
SINGLE LINKED LIST

Disusun Oleh:

Muhammad Shafiq Rasuna - 2311104043

Kelas :

S1SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd, M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

1. Memahami penggunaan linked list dengan pointer operator-operator dalam program
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada

2. Landasan Teori

2.1. Linked List dengan Pointer

Linked list (biasa disebut list saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam Linked list bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe string. Sedangkan data majemuk merupakan sekumpulan data (record) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe string, NIM bertipe long integer, dan Alamat bertipe string.

Linked list dapat diimplementasikan menggunakan Array dan Pointer (Linked list). Yang akan kita gunakan adalah pointer, karena beberapa alasan, yaitu :

1. Array bersifat statis, sedangkan pointer dinamis.
2. Pada linked list bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai pointer.
3. Sifat linked list yang fleksibel lebih cocok dengan sifat pointer yang dapat diatur sesuai kebutuhan.
4. Karena array lebih susah dalam menangani linked list, sedangkan pointer lebih mudah.
5. Array lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.

Dalam implementasinya, pengaksesan elemen pada Linked list dengan pointer bisa menggunakan (->) atau tanda titik (.).

Model-model dari ADT Linked list yang kita pelajari adalah :

1. Single Linked list
2. Double Linked list
3. Circular Linked list
4. Multi Linked list

5. Stack (Tumpukan)
6. Queue (Antrian)
7. Tree
8. Graph

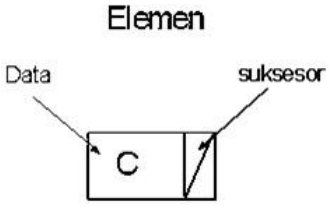
Setiap model ADT Linked list di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan.
Secara umum operasi-operasi ADT pada Linked list, yaitu :

1. Penciptaan dan inisialisasi list (Create List).
2. Penyisipan elemen list (Insert).
3. Penghapusan elemen list (Delete).
4. Penelusuran elemen list dan menampilkannya (View).
5. Pencarian elemen list (Searching).
6. Pengubahan isi elemen list (Update).

2.2 Single Linked List

Single Linked list merupakan model ADT Linked list yang hanya memiliki satu arah pointer.

Komponen elemen dalam single linked list:



Gambar 4-1 Elemen *Single Linked list*

Keterangan:
Elemen: segmen-segmen data yang terdapat dalam suatu *list*.
Data: informasi utama yang tersimpan dalam sebuah elemen.
Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari *Single Linked list*:

1. Hanya memerlukan satu buah *pointer*.
2. *Node* akhir menunjuk ke Nil kecuali untuk *list circular*.
3. Hanya dapat melakukan pembacaan maju.
4. Pencarian sequensial dilakukan jika data tidak terurut.
5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah *list*.

Istilah-istilah dalam *Single Linked list* :

1. *first*/*head*: *pointer* pada *list* yang menunjuk alamat elemen pertama *list*.
2. *next*: *pointer* pada elemen yang berfungsi sebagai *successor* (penunjuk) alamat elemen di depannya.

3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. *Node*/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Contoh deklarasi struktur data *single linked list*:

```
1  /*file : list.h*/
2  #ifndef LIST_H_INCLUDED
3  #define LIST_H_INCLUDED
4
5  #define Nil NULL
6  #define info(P) (P)->info
7  #define next(P) (P)->next #define
8  first(L) ((L).first) using
9  namespace std;
10 /*deklarasi record dan struktur data
11 list*/ typedef int infotype; typdef struct
12 elmlist *address; struct elmlist {
13 infotype info;    address next;
14 };
15 struct list{
16 address first;
17 };
18 #endif // TEST_H_INCLUDED
19
20
21
```

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
1  /*file : list.h*/
2  #ifndef LIST_H_INCLUDED
3  #define LIST_H_INCLUDED
4
5      #define Nil NULL
6      #define info(P) (P)->info
7      #define next(P) (P)->next
8      #define first(L) ((L).first)
9
10     using namespace std;
11     /*deklarasi record dan struktur data list*/
12 struct mahasiswa{    char nama[30]    char nim[10]
13 }
14 typedef mahasiswa infotype;
15
16 typdef struct elmlist
17 *address; struct elmlist {
18 infotype info;    address
19 next;
20 };
21 struct list{
22 address first;
23 };
24 #endif // TEST_H_INCLUDED
25
26
27
```

Pembentukan Komponen-Komponen *List*

A. Pembentukan *List*

Adalah sebuah proses untuk membetuk sebuah *list* baru. Biasanya nama fungsi yang digunakan **createList().** Fungsi ini akan mengeset nilai awal *list* yaitu *first*(*list*) dan *last*(*list*) dengan nilai Nil.

B. Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam *list*. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan alokasi().

Sintak alokasi pada C:

P = (*address*) malloc (sizeof (elmlist));

Keterangan:

P	= variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address	= tipe data *pointer* dari tipe data elemen yang akan dialokasikan.
Elmlist	= tipe data atau *record* elemen yang dialokasikan.

Contoh deklarasi struktur data *single linked list*:
Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
address p =
(address)malloc(sizeof(elmlist))
;      info(p) = m;      return
p;
}
```

Namun pada Cpp. Penggunaan **malloc** dapat dipersingkat menggunakan sintak **new.**

Sintak alokasi pada Cpp:

P = new elmlist;

Keterangan:
P = variabel *pointer* yang mengacu pada elemen yang dialokasikan. *address* = tipe data *pointer* dari tipe data elemen yang akan dialokasikan.
Contoh deklarasi struktur data *single linked list*:
Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
    address p = new elmlist;
    info(p) = m;
    return p;
}
```

C. *Dealokasi*

Untuk menghapus sebuah *memory address* yang tersimpan atau telah dialokasikan dalam bahasa pemrograman C digunakan sintak ***free***, sedangkan pada Cpp digunakan sintak ***delete***, seperti berikut.

Sintak pada C:
free(p);
Sintak pada Cpp:
delete p;

D. *Pengecekan List*

Adalah fungsi untuk mengecek apakah *list* tersebut kosong atau tidak. Akan mengembalikan nilai true jika *list* kosong dan nilai *false* jika *list* tidak kosong. Fungsi yang digunakan adalah isEmpty()

3. Guided

1. SINGLE LINKED LIST

Program ini adalah contoh sederhana untuk memahami dan mengelola data menggunakan struktur data linked list. Penggunaan fungsi-fungsi ini juga mencerminkan dasar-dasar pengelolaan memori dinamis di C++.

Kode program :

Telkom University Purwokerto

```cpp
#include <iostream>
#include <cstring>
using namespace std;

// Deklarasi Struct untuk mahasiswa
struct mahasiswa {
    char nama[30];
    char nim[10];
};

// Deklarasi Struct Node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// Inisialisasi List
void init() {
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == nullptr;
}

// Tambah Depan
void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != nullptr) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = nullptr; // List menjadi kosong
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            delete tail;
            tail = bantu;
            tail->next = nullptr;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Tampilkan List
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        while (current != nullptr) {
            cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus List
void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
    init();

    // Contoh data mahasiswa
    mahasiswa m1 = {"Alice", "123456"};
    mahasiswa m2 = {"Bob", "654321"};
    mahasiswa m3 = {"Charlie", "112233"};

    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

    // Menghapus elemen dari list
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

Output program :

```
muan4\"guided1
Nama: Alice, NIM: 123456
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
List berhasil terhapus!

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\
```

2. SINGLE LINK LIST

Kode C++ di atas mengimplementasikan linked list sederhana untuk menyimpan dan mengelola data integer. Struktur utama dari linked list adalah Node, yang terdiri dari dua atribut: data, untuk menyimpan nilai integer, dan next, sebagai pointer yang menunjuk ke elemen berikutnya dalam list. Fungsi utama yang didefinisikan dalam program ini meliputi alokasi dan dealokasi memori untuk node, serta beberapa operasi dasar pada linked list. Fungsi alokasi digunakan untuk membuat node baru, sedangkan dealokasi bertanggung jawab untuk mengembalikan memori yang digunakan. Untuk mengelola elemen dalam list, terdapat fungsi untuk menambahkan elemen di awal (insertFirst) dan di akhir (insertLast), memeriksa apakah list kosong (isListEmpty), serta menampilkan elemen (printList) dan menghitung jumlah elemen (countElements). Program ini juga mencakup fungsi untuk menghapus semua elemen dalam list melalui clearList, memastikan bahwa semua memori yang digunakan dibebaskan. Dalam fungsi main, program dimulai dengan membuat list kosong dan kemudian melakukan serangkaian operasi: menambahkan elemen, menampilkan isi list, menghitung jumlah elemen, dan menghapus seluruh list, dengan hasil output yang menunjukkan status list pada setiap langkah. Secara keseluruhan, program ini memberikan contoh praktis tentang pengelolaan data menggunakan struktur linked list di C++.

Telkom University Purwokerto

```cpp
#include <iostream>
using namespace std;

// Definisi struktur untuk elemen list
struct Node {
    int data;        // Menyimpan nilai elemen
    Node* next;      // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* alokasi(int value) {
    Node* newNode = new Node; // Alokasi memori untuk elemen baru
    if (newNode != nullptr) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = nullptr; // Set next ke nullptr
    }
    return newNode; // Mengembalikan pointer node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori yang digunakan oleh node
}

// Pengecekan apakah list kosong
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}

// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        newNode->next = head;      // Menghubungkan elemen baru ke elemen pertama
        head = newNode;            // Menetapkan elemen baru sebagai elemen pertama
    }
}

// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != nullptr) { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        cout << endl;
    }
}

// Menghitung jumlah elemen dalam list
int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;             // Menambah jumlah elemen
        temp = temp->next; // Melanjutkan ke elemen berikutnya
    }
    return count;            // Mengembalikan jumlah elemen
}

// Menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node* &head) {
    while (head != nullptr) {
        Node* temp = head;  // Simpan pointer ke node saat ini
        head = head->next;  // Pindahkan ke node berikutnya
        dealokasi(temp);    // Dealokasi node
    }
}

int main() {
    Node* head = nullptr;  // Membuat list kosong

    // Menambahkan elemen ke dalam list
    insertFirst(head, 10); // Menambahkan elemen 10 di awal list
    insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
    insertLast(head, 30);  // Menambahkan elemen 30 di akhir list

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

    // Menampilkan jumlah elemen
    cout << "Jumlah elemen: " << countElements(head) << endl;

    // Menghapus semua elemen dalam list
    clearList(head);

    // Menampilkan isi list setelah penghapusan
    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

Output program :

```
c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan4>cd "c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman
 Struktur Data 3\pertemuan4\" && g++ guided.2.cpp -o guided.2 && "c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\per
temuan4\"guided.2
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan4>
```

4. Unguided

4.1. Kode program :

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data ;
       if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}


int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked List: ";
    printList(head);

    return 0;
}
```

Output dari kode program :

```
c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan4>cd "c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman
 Struktur Data 3\pertemuan4\" && g++ unguided1.cpp -o unguided1 && "c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\p
ertemuan4\"unguided1
Linked List: 5 -> 10 -> 20
```

4.2. Kode program :

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

// Function to insert a node at the front
void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a node at the back
void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to delete a node with a given value
void deleteNode(Node** head, int value) {
    Node* temp = *head;
    Node* prev = nullptr;

    // If head node itself holds the value to be deleted
    if (temp != nullptr && temp->data == value) {
        *head = temp->next; // Change head
        delete temp; // Free old head
        return;
    }

    // Search for the value to be deleted
    while (temp != nullptr && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

    // If value was not present in the list
    if (temp == nullptr) return;

    // Unlink the node from linked list
    prev->next = temp->next;

    delete temp; // Free memory
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        temp = temp->next;
        if (temp != nullptr) {
            cout << " -> ";
        }
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked List before deletion: ";
    printList(head);

    deleteNode(&head, 10);

    cout << "Linked List after deletion: ";
    printList(head);

    return 0;
}
```

Output dari kode program :

```
c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan4>cd "c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman
 Struktur Data 3\pertemuan4\" && g++ unguided2.cpp -o unguided2 && "c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\p
ertemuan4\"unguided2
Linked List before deletion: 5 -> 10 -> 20
Linked List after deletion: 5 -> 20

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan4>
```

4.3 kode program :

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

// Function to insert a node at the front
void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a node at the back
void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to search for a node with a given value
bool searchNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

// Function to count the length of the linked list
int countLength(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        temp = temp->next;
        if (temp != nullptr) {
            cout << " -> ";
        }
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked List: ";
    printList(head);

    // Search for a value
    int searchValue = 20;
    if (searchNode(head, searchValue)) {
        cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
    }

    // Count length of the list
    int length = countLength(head);
    cout << "Panjang linked list: " << length << endl;

    return 0;
}
```

Maka Akan Menghasilkan Output Seperti Dibawah :

```
Linked List: 5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan4>
```

5. Kesimpulan

Kesimpulan dari laporan praktikum ini adalah bahwa penggunaan linked list sebagai struktur data memberikan fleksibilitas dalam pengelolaan elemen data dibandingkan dengan array. Melalui implementasi single linked list, peserta praktik dapat memahami konsep dasar dari pengelolaan memori dinamis, termasuk alokasi dan dealokasi memori menggunakan pointer. Praktikum ini juga menunjukkan operasi dasar pada linked list, seperti penyisipan, penghapusan, dan pencarian elemen, yang dapat dilakukan dengan efisien. Selain itu, peserta praktik belajar tentang cara mengakses elemen menggunakan operator pointer, serta pentingnya fungsi pengecekan untuk menentukan apakah list kosong atau tidak. Dengan demikian, praktikum ini memberikan landasan yang kuat untuk memahami konsep-konsep lanjutan dalam pengelolaan data dan algoritma, serta aplikasi nyata dari struktur data linked list dalam pemrograman C++.

.