LAPORAN PRAKTIKUM

Modul 04

“SINGLE LINKED LIST (BAGIAN PERTAMA)”

Disusun Oleh:

Dimas Abhipraya Ramansyah

(2311104053)SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

Telkom University Purwokerto

1. Tujuan

a) Membangun program yang memanfaatkan struktur data linked list.

b) Belajar memprogram dengan linked list.

c) Mempelajari dan menerapkan linked list dalam pemrograman.

2. Landasan Teori

A. Link list

Linked list, atau yang sering disebut sebagai list, adalah struktur data yang terdiri dari serangkaian elemen yang saling terhubung dan fleksibel, memungkinkan penambahan atau pengurangan elemen sesuai kebutuhan.

B. Single link list

Single Linked List adalah model dari Abstract Data Type (ADT) linked list yang memiliki satu arah pointer. Dalam struktur ini, setiap elemen terdiri dari beberapa komponen, yaitu data yang menyimpan informasi utama, dan suksesor yang berfungsi sebagai penghubung antar elemen.

1. Pembentukan Komponen – Komponen List :
    a. Pembentuk List : Proses membuat list baru menggunakan fungsi create list, yang mengatur (first list) dan last (list) ke Nil.
    b. Pengalokasian Memori : Proses untuk mengalokasikan memori bagi elemen list dengan fungsi alokasi().
    c. Dealokasi : Menghapus memori yang dialokasikan. Dalam C, gunakan free(p);, dan dalam C++ gunakan delete;.
    d. Pengecekan List : Fungsi isEmpty() untuk memeriksa apakah list kosong. Mengembalikan true jika kosong, false jika tidak.
2. Insert
    a. Insert First : Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada awal list.
    b. Insert Last : Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada akhir list.
    c. Insert After : Merupakan metode memasukkan data ke dalam list yang diletakkan setelah node tertentu yang ditunjuk oleh user.
3. View

    Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut.

C. Delete

1. Delete First
   Adalah pengambilan atau penghapusan sebuah elemen pada awal list.
2. Delete Last
   Merupakan pengambilan atau penghapusan suatu elemen dari akhir list.
3. Delate After
   Merupakan pengambilan atau penghapusan node setelah node tertentu.

4. Delete Elemen

   Operasi yang digunakan untuk menghapus dan membebaskan memori yang digunakan oleh elemen dalam list disebut dealokasi. Terdapat dua fungsi utama yang sering digunakan dalam proses ini, yaitu fungsi dealokasi(P), yang berfungsi untuk membebaskan memori yang digunakan oleh elemen P, dan fungsi delAll(L), yang membebaskan semua memori yang digunakan oleh elemen-elemen dalam list L, sehingga list L menjadi kosong.

D. Update

Merupakan operasi dasar pada list yang digunakan untuk meng-update data yang ada di dalam list. Dengan operasi update ini kita dapat meng-update data-data node yang ada di dalam list. Proses update biasanya diawali dengan proses pencarian terhadap data yang akan di-update.

Kelebihan Linked List:

- Fleksibel: Mudah menambah atau menghapus elemen.
- Dinamis: Ukurannya dapat berubah sesuai kebutuhan.

Struktur Node:

- Data: Menyimpan informasi yang ingin disimpan.
- Pointer: Menunjuk ke node berikutnya.

Jenis-jenis Linked List:

- Single Linked List: Setiap node hanya memiliki satu pointer ke node berikutnya.
- Double Linked List: Setiap node memiliki dua pointer, ke node sebelumnya dan berikutnya.
- Circular Linked List: Node terakhir menunjuk ke node pertama.

Operasi Dasar:

- Create: Membuat list baru.
- Insert: Menambahkan node ke list.
- Delete: Menghapus node dari list.
- Search: Mencari node tertentu.
- Traverse: Melalui semua node dalam list.

Implementasi:

- Menggunakan pointer untuk menghubungkan node-node.
- Alokasi memori dinamis untuk setiap node baru.

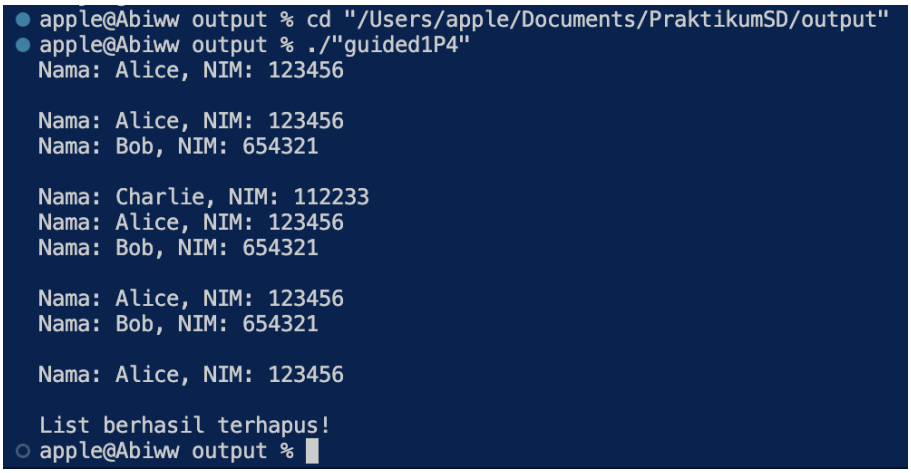
Contoh Penggunaan:

- Menyimpan data yang ukurannya tidak tetap.
- Mengimplementasikan struktur data lain seperti stack, queue, tree, dan graph.

Telkom University Purwokerto

3. Guided

```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;
    }
    cout << 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;
}
```



Cara kerja program:

1. Deklarasi struktur: Program mendefinisikan struktur mahasiswa untuk menyimpan data mahasiswa dan struktur Node untuk mewakili setiap node dalam linked list.
2. Inisialisasi: Fungsi init() membuat linked list kosong.
3. Operasi: Program melakukan berbagai operasi pada linked list, seperti menambahkan, menghapus, dan menampilkan data.
4. Contoh penggunaan: Program memberikan contoh cara menggunakan fungsi-fungsi yang telah didefinisikan.

```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;
}
```

```
apple@Abiww output % cd "/Users/apple/Documents/PraktikumSD/output"
apple@Abiww output % ./"guided2P4"
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
apple@Abiww output %
```

Cara kerja:

1. Struktur Node: Setiap node memiliki dua anggota: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya.
2. Operasi: Program menyediakan berbagai operasi pada linked list, seperti menambahkan, menghapus, dan menampilkan elemen.
3. Alokasi dan dealokasi memori: Fungsi alokasi dan dealokasi digunakan untuk mengelola memori secara efisien.
4. Pointer head: Pointer head menunjuk ke node pertama dalam list.

4. Unguided

1. Membuat Single Linked List.

```cpp
//Membuat Single Linked List

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

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

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

    if (head == nullptr) {
        head = newNode;
    } else {
        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;

    insertAtFront(head, 10);
    insertAtBack(head, 20);
    insertAtFront(head, 5);

    printList(head);

    return 0;
}
```

```
apple@Abiww output % cd "/Users/apple/Documents/PraktikumSD/output"
apple@Abiww output % ./"unguided4"
5 -> 10 -> 20
apple@Abiww output %
```

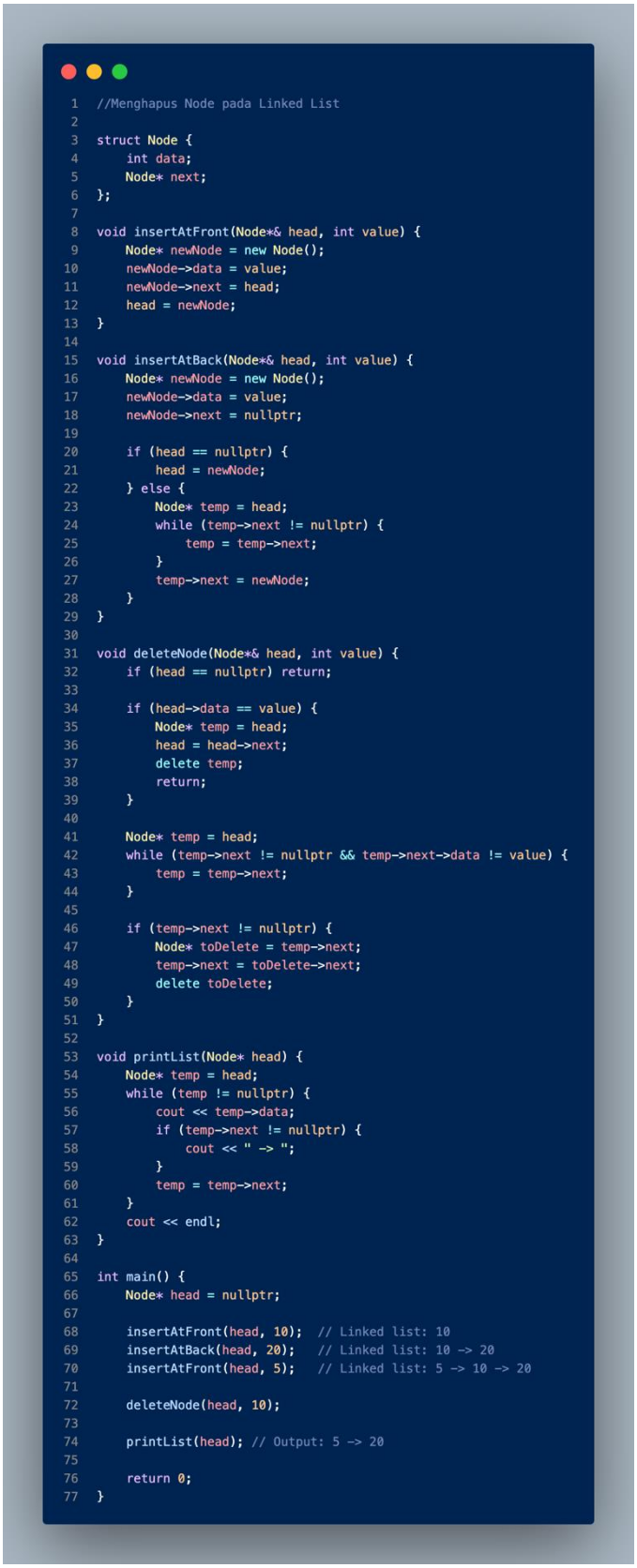
Fungsi-fungsi utama:

- insertDepan: Menambahkan node di awal linked list.
- insertBelakang: Menambahkan node di akhir linked list.
- cetakList: Mencetak semua elemen dalam linked list.

Cara kerja:

1) Struktur Node: Setiap node memiliki dua anggota: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya.
2) Operasi: Program menyediakan fungsi untuk menambahkan node di awal dan akhir linked list.
3) Pointer head: Pointer head menunjuk ke node pertama dalam linked list.

2. Menghapus Node pada Linked List



```cpp
//Menghapus Node pada Linked List

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

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

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

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

void deleteNode(Node*& head, int value) {
    if (head == nullptr) return;

    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

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

    if (temp->next != nullptr) {
        Node* toDelete = temp->next;
        temp->next = toDelete->next;
        delete toDelete;
    }
}

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;

    insertAtFront(head, 10);  // Linked list: 10
    insertAtBack(head, 20);   // Linked list: 10 -> 20
    insertAtFront(head, 5);   // Linked list: 5 -> 10 -> 20

    deleteNode(head, 10);

    printList(head); // Output: 5 -> 20

    return 0;
}
```

Fungsi-fungsi utama:

- insertDepan: Menambahkan node di awal linked list.
- insertBelakang: Menambahkan node di akhir linked list.
- hapusNode: Menghapus node dengan nilai tertentu dari linked list.
- cetakList: Mencetak semua elemen dalam linked list.

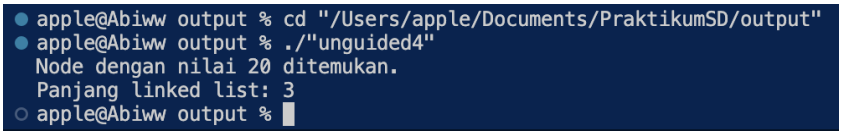
Cara Kerja:

1) Struktur Node: Setiap node memiliki dua anggota: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya.
2) Operasi: Program menyediakan fungsi untuk menambahkan, menghapus, dan menampilkan elemen dalam linked list.
3) Pointer head: Pointer head menunjuk ke node pertama dalam linked list.

Telkom University Purwokerto

3. Mencari dan Menghitung Panjang Linked List

```cpp
//Mencari dan Menghitung Panjang Linked List

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

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

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

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

bool searchNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

int lengthOfList(Node* head) {
    int length = 0;
    Node* temp = head;
    while (temp != nullptr) {
        length++;
        temp = temp->next;
    }
    return length;
}

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;

    insertAtFront(head, 10);
    insertAtBack(head, 20);
    insertAtFront(head, 5);

    if (searchNode(head, 20)) {
        cout << "Node dengan nilai 20 ditemukan." << endl;
    } else {
        cout << "Node dengan nilai 20 tidak ditemukan." << endl;
    }

    cout << "Panjang linked list: " << lengthOfList(head) << endl; // Output: 3

    return 0;
}
```



Fungsi-fungsi dasar:

- Menambahkan node: Fungsi insertDepan dan insertBelakang untuk menambahkan node di awal dan akhir list.
- Mencetak list: Fungsi cetakList untuk menampilkan semua data dalam list.
- Mencari node: Fungsi cariNode untuk mencari keberadaan suatu nilai dalam list.
- Menghitung panjang: Fungsi hitungPanjang untuk menghitung jumlah node dalam list.

Cara Kerja:

1) Struktur Node: Setiap node memiliki data dan pointer ke node berikutnya.
2) Pointer head: Menunjuk ke node pertama dalam list.
3) Operasi: Fungsi-fungsi di atas melakukan operasi pada linked list, seperti menambahkan, mencari, dan menghitung node.

5. Kesimpulan

Linked list adalah struktur data dinamis yang fleksibel, memungkinkan penambahan dan penghapusan elemen dengan efisien menggunakan pointer. Salah satu modelnya adalah single linked list, terdiri dari node yang saling terhubung satu arah, di mana setiap node memiliki data dan pointer yang menunjuk ke node berikutnya, sementara node terakhir menunjuk ke nilai null. Operasi dasar pada Single Linked List meliputi penambahan elemen di awal, akhir, atau setelah node tertentu (insert), serta penghapusan elemen pertama, terakhir, atau setelah node tertentu (delete). Selain itu, linked list mendukung pengecekan apakah list kosong, pencarian elemen, dan pembaruan data (update). Kelebihan Single Linked List adalah kemudahan dalam penambahan dan penghapusan elemen, meskipun pencarian harus dilakukan secara sekuensial.