

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN  
ALGORITMA**

**MODUL IV  
LINKED LIST CIRCULAR DAN NON CIRCULAR**



**Disusun Oleh :**

Raka Andriy Shevchenko  
2311102054

**Dosen**

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

**PROGRAM STUDI S1 TEKNIK INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

## A. Dasar Teori

Linked List adalah sebuah struktur data dinamis yang digunakan untuk menyimpan dan mengelola data secara dinamis. Dalam Linked List, data disimpan dalam bentuk simpul atau node yang saling terhubung satu sama lain dengan menggunakan referensi atau alamat. Setiap simpul berisi dua hal penting: data yang ingin disimpan dan alamat referensi ke simpul berikutnya dalam urutan. Dengan begitu, setiap simpul dapat menyimpan data dan mengetahui alamat simpul selanjutnya. Setiap Linked List memiliki dua elemen khusus, yaitu "head" dan "tail":

- Head: Merupakan simpul pertama dalam Linked List dan berfungsi sebagai titik awal akses ke seluruh data dalam Linked List.
- Tail: Merupakan simpul terakhir dalam Linked List dan menjadi penanda akhir dari urutan simpul.

Linked List memiliki beberapa fungsi penting, antara lain:

- Menyimpan dan mengelola data dalam urutan tertentu.
- Memudahkan penambahan dan penghapusan data secara dinamis tanpa harus menggeser data lain.
- Digunakan dalam implementasi berbagai algoritma dan struktur data lain seperti stack dan queue.

Linked List dapat dibagi menjadi beberapa jenis, antara lain:

- Non-Circular Linked List adalah sebuah jenis Linked List yang tidak memiliki hubungan dengan simpul pertama atau terakhir dalam urutan. Dalam Non-circular Linked List, setiap simpul hanya memiliki referensi ke simpul berikutnya dalam urutan. Setiap simpul dapat menyimpan data dan mengetahui alamat simpul selanjutnya.
- Circular Linked List adalah sebuah jenis Linked List yang memiliki hubungan yang berlanjung dengan simpul pertama dan terakhir dalam urutan. Dalam Circular Linked List, setiap simpul memiliki referensi ke simpul berikutnya dalam urutan, sama seperti pada Linked List biasa. Namun, pada Circular Linked List, referensi terakhir dari simpul berikutnya akan menuju ke simpul

pertama dalam Linked List. Setiap simpul dapat menyimpan data dan mengetahui alamat simpul selanjutnya.

## B. Guided

### a. Guided 1

Source Code:

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

// Definisi struktur Node untuk Linked List
struct Node {
    int data;           // Data yang disimpan dalam node
    Node *next;         // Pointer yang menunjukkan ke node
                        // berikutnya
};

Node *head; // Pointer ke kepala/awal linked list
Node *tail; // Pointer ke ekor/akhir linked list

// Fungsi untuk menginisialisasi linked list
void init() {
    head = NULL; // Atur pointer head menjadi NULL
    tail = NULL; // Atur pointer tail menjadi NULL
}

// Fungsi untuk memeriksa apakah linked list kosong
bool isEmpty() {
    return head == NULL; // Mengembalikan true jika head
                        // adalah NULL
}

// Fungsi untuk menyisipkan node baru di depan linked list
void insertDepan(int nilai) {
    Node *baru = new Node; // Buat node baru
    baru->data = nilai;     // Atur nilai data pada node baru
    baru->next = NULL;      // Atur pointer next node baru
                        // menjadi NULL

    if (isEmpty()) {        // Jika linked list kosong
        head = tail = baru; // Atur head dan tail menjadi
                        // node baru
    } else {
        baru->next = head;  // Atur pointer next node baru
                        // menuju head
    }
}
```

```

        head = baru;          // Perbarui head menjadi node
baru
    }
}

// Fungsi untuk menyisipkan node baru di belakang linked list
void insertBelakang(int nilai) {
    Node *baru = new Node;    // Buat node baru
    baru->data = nilai;        // Atur nilai data pada node baru
    baru->next = NULL;         // Atur pointer next node baru
menjadi NULL

    if (isEmpty()) {          // Jika linked list kosong
        head = tail = baru;   // Atur head dan tail menjadi
node baru
    } else {
        tail->next = baru;     // Atur pointer next tail menuju
node baru
        tail = baru;          // Perbarui tail menjadi node
baru
    }
}

// Fungsi untuk menghitung jumlah node dalam linked list
int hitungList() {
    Node *hitung = head;     // Pointer untuk menelusuri linked
list
    int jumlah = 0;           // Variabel untuk menyimpan jumlah
node

    while (hitung != NULL) {   // Selama pointer tidak NULL
        jumlah++;              // Tambahkan 1 ke jumlah
        hitung = hitung->next;  // Geser pointer ke node
berikutnya
    }
    return jumlah;             // Kembalikan jumlah node
}

// Fungsi untuk menyisipkan node baru di tengah linked list
pada posisi tertentu
void insertTengah(int data, int posisi) {
    // Penanganan jika posisi di luar jangkauan atau posisi
adalah 1

```

```

if (posisi < 1 || posisi > hitungList()) {
    cout << "Posisi diluar jangkauan" << endl;
} else if (posisi == 1) {
    cout << "Posisi bukan posisi tengah" << endl;
} else {
    Node *baru = new Node(); // Buat node baru
    baru->data = data;        // Atur nilai data pada
node baru
    Node *bantu = head;      // Pointer bantu untuk
menelusuri linked list
    int nomor = 1;           // Nomor untuk menunjukkan
posisi saat ini

    // Temukan posisi sebelum posisi yang dituju
    while (nomor < posisi - 1) {
        bantu = bantu->next; // Geser pointer bantu
        nomor++;             // Tambahkan 1 ke nomor
    }

    // Sisipkan node baru di antara node yang tepat
    baru->next = bantu->next; // Atur pointer next node
baru
    bantu->next = baru;       // Atur pointer next node
sebelumnya
    }
}

// Fungsi untuk menghapus node dari depan linked list
void hapusDepan() {
    // Penanganan jika linked list tidak kosong
    if (!isEmpty()) {
        Node *hapus = head; // Simpan pointer ke node
yang akan dihapus

        // Jika masih ada node lain setelah head
        if (head->next != NULL) {
            head = head->next; // Atur head baru menjadi
node setelahnya
        } else {
            head = tail = NULL; // Atur head dan tail
menjadi NULL
        }
        delete hapus; // Hapus node yang disimpan
    }
}

```

```

    } else {
        cout << "List kosong!" << endl; // Tampilkan pesan
        jika linked list kosong
    }
}

// Fungsi untuk menghapus node dari belakang linked list
void hapusBelakang() {
    // Penanganan jika linked list tidak kosong
    if (!isEmpty()) {
        Node *hapus = tail; // Simpan pointer ke node
        yang akan dihapus

        // Jika linked list memiliki lebih dari satu node
        if (head != tail) {
            Node *bantu = head; // Pointer bantu untuk
            menelusuri linked list
            // Temukan node sebelum tail
            while (bantu->next != tail) {
                bantu = bantu->next; // Geser pointer bantu
            }
            tail = bantu; // Perbarui tail menjadi
            node sebelumnya
            tail->next = NULL; // Atur pointer next tail
            menjadi NULL
        } else {
            head = tail = NULL; // Atur head dan tail
            menjadi NULL
        }
        delete hapus; // Hapus node yang
        disimpan
    } else {
        cout << "List kosong!" << endl; // Tampilkan pesan
        jika linked list kosong
    }
}

// Fungsi untuk menghapus node dari tengah linked list pada
posisi tertentu
void hapusTengah(int posisi) {
    // Penanganan jika posisi di luar jangkauan atau posisi
    adalah 1
    if (posisi < 1 || posisi > hitungList()) {

```

```

        cout << "Posisi di luar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    } else {
        Node *bantu = head;    // Pointer bantu untuk
menelusuri linked list
        Node *hapus;          // Pointer untuk menyimpan
node yang akan dihapus
        Node *sebelum = NULL; // Pointer untuk menyimpan
node sebelum node yang akan dihapus
        int nomor = 1;        // Nomor untuk menunjukkan
posisi saat ini

        // Temukan node sebelum node yang akan dihapus
        while (nomor < posisi) {
            sebelum = bantu;    // Simpan pointer sebelum
            bantu = bantu->next; // Geser pointer bantu
            nomor++;            // Tambahkan 1 ke nomor
        }

        hapus = bantu;          // Simpan pointer ke node yang
akan dihapus
        if (sebelum != NULL) {
            sebelum->next = bantu->next; // Hubungkan node
sebelumnya dengan node setelahnya
        } else {
            head = bantu->next;    // Atur head baru jika node
pertama dihapus
        }
        delete hapus;            // Hapus node yang
disimpan
    }
}

// Fungsi untuk mengubah nilai data dari node di depan linked
list
void ubahDepan(int data) {
    // Penanganan jika linked list tidak kosong
    if (!isEmpty()) {
        head->data = data; // Ubah nilai data pada node head
    } else {
        cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
    }
}

```



```

    }
}

// Fungsi untuk mengubah nilai data dari node di tengah linked
// list pada posisi tertentu
void ubahTengah(int data, int posisi) {
    // Penanganan jika linked list tidak kosong
    if (!isEmpty()) {
        // Penanganan jika posisi di luar jangkauan atau
        // posisi adalah 1
        if (posisi < 1 || posisi > hitungList()) {
            cout << "Posisi di luar jangkauan" << endl;
        } else if (posisi == 1) {
            cout << "Posisi bukan posisi tengah" << endl;
        } else {
            Node *bantu = head; // Pointer bantu untuk
            // menelusuri linked list
            int nomor = 1;      // Nomor untuk menunjukkan
            // posisi saat ini

            // Temukan node pada posisi yang dituju
            while (nomor < posisi) {
                bantu = bantu->next; // Geser pointer bantu
                nomor++;             // Tambahkan 1 ke nomor
            }

            bantu->data = data; // Ubah nilai data pada
            // node yang dituju
        }
    } else {
        cout << "List masih kosong!" << endl; // Tampilkan
        // pesan jika linked list kosong
    }
}

// Fungsi untuk mengubah nilai data dari node di belakang
// linked list
void ubahBelakang(int data) {
    // Penanganan jika linked list tidak kosong
    if (!isEmpty()) {
        tail->data = data; // Ubah nilai data pada node tail
    } else {

```

```

        cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
    }
}

// Fungsi untuk menghapus semua node dari linked list
void clearList() {
    Node *bantu = head; // Pointer bantu untuk menelusuri
linked list
    Node *hapus;        // Pointer untuk menyimpan node yang
akan dihapus

    // Selama masih ada node yang tersisa
    while (bantu != NULL) {
        hapus = bantu;    // Simpan pointer ke node yang akan
dihapus
        bantu = bantu->next; // Geser pointer bantu ke node
berikutnya
        delete hapus;      // Hapus node yang disimpan
    }

    head = tail = NULL; // Atur head dan tail menjadi NULL
    cout << "List berhasil terhapus!" << endl; // Tampilkan
pesan berhasil
}

// Fungsi untuk menampilkan isi linked list
void tampil() {
    Node *bantu = head; // Pointer untuk menelusuri linked
list

    if (!isEmpty()) {    // Jika linked list tidak kosong
        while (bantu != NULL) {
            cout << bantu->data << " "; // Tampilkan nilai
data node
            bantu = bantu->next;        // Geser pointer ke
node berikutnya
        }
        cout << endl; // Pindah baris setelah selesai
menampilkan linked list
    } else {
        cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
    }
}

```

```

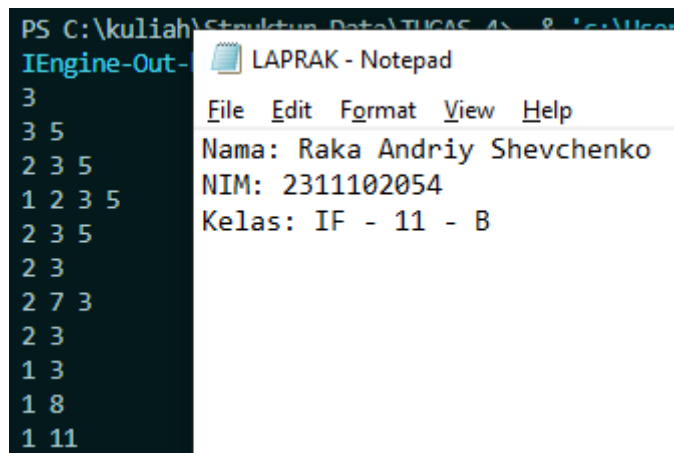
    }
}

// Fungsi utama
int main() {
    init();           // Inisialisasi linked list
    insertDepan(3);   // Sisipkan node dengan nilai 3 di
    // depan linked list
    tampil();         // Tampilkan isi linked list
    insertBelakang(5); // Sisipkan node dengan nilai 5 di
    // belakang linked list
    tampil();
    insertDepan(2);   // Sisipkan node dengan nilai 2 di
    // depan linked list
    tampil();
    insertDepan(1);   // Sisipkan node dengan nilai 1 di
    // depan linked list
    tampil();
    hapusDepan();     // Hapus node pertama dari linked list
    tampil();
    hapusBelakang();  // Hapus node terakhir dari linked list
    tampil();
    insertTengah(7, 2); // Sisipkan node dengan nilai 7 di
    // posisi kedua linked list
    tampil();
    hapusTengah(2);   // Hapus node pada posisi kedua dari
    // linked list
    tampil();
    ubahDepan(1);     // Ubah nilai data node pertama
    // menjadi 1
    tampil();
    ubahBelakang(8);  // Ubah nilai data node terakhir
    // menjadi 8
    tampil();
    ubahTengah(11, 2); // Ubah nilai data node pada posisi
    // kedua menjadi 11
    tampil();

    return 0;
}

```

Output:



```
PS C:\kuliah\Struktur Data\TUGAS 4> g++ 1.cpp & .\IEngine-Output.exe
3
3 5
2 3 5
1 2 3 5
2 3 5
2 3
2 7 3
2 3
1 3
1 8
1 11
```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

Deskripsi:

Program C++ di atas mengimplementasikan operasi dasar pada linked list seperti penyisipan di depan, di belakang, dan di tengah, penghapusan dari depan, belakang, dan tengah, serta pengubahan nilai data pada node tertentu. Karena penjelasan detail sudah ada di source code, jadi saya akan menjelaskan alur programnya. Alur program dimulai dengan inisialisasi linked list, di mana beberapa node disisipkan di depan dan di belakang. Setiap fungsi memastikan keadaan linked list dan menangani penyesuaian pointer dengan benar. Program kemudian menampilkan isi linked list setelah setiap operasi yang dilakukan.

b. Guided 2

Source Code:

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

// Definisi struktur Node untuk Linked List
struct Node {
    string data; // Data yang disimpan dalam node
    Node *next; // Pointer yang menunjukkan ke node
    berikutnya
};
```

```

// Deklarasi pointer global untuk head, tail, serta node baru,
bantu, dan hapus
Node *head, *tail, *baru, *bantu, *hapus;

// Fungsi untuk menginisialisasi linked list
void init() {
    head = NULL; // Atur pointer head menjadi NULL
    tail = head; // Atur pointer tail menjadi NULL
}

// Fungsi untuk memeriksa apakah linked list kosong
int isEmpty() {
    return head == NULL; // Mengembalikan 1 jika head adalah
NULL, 0 jika tidak
}

// Fungsi untuk membuat node baru dengan data tertentu
void buatNode(string data) {
    baru = new Node; // Buat node baru
    baru->data = data; // Atur nilai data pada node baru
    baru->next = NULL; // Atur pointer next node baru menjadi
NULL
}

// Fungsi untuk menghitung jumlah node dalam linked list
int hitungList() {
    bantu = head; // Pointer untuk menelusuri linked list
    int jumlah = 0; // Variabel untuk menyimpan jumlah node

    while (bantu != NULL) { // Selama pointer tidak NULL
        jumlah++; // Tambahkan 1 ke jumlah
        bantu = bantu->next; // Geser pointer ke node
berikutnya
    }
    return jumlah; // Kembalikan jumlah node
}

// Fungsi untuk menyisipkan node baru di depan linked list
void insertDepan(string data) {
    buatNode(data); // Buat node baru dengan data yang
diberikan
    if (isEmpty()) { // Jika linked list kosong
        head = baru; // Atur head menjadi node baru
    }
}

```

```

        tail = head; // Atur tail menjadi head
        baru->next = head; // Atur pointer next node baru ke
head
    } else {
        while (tail->next != head) { // Temukan node terakhir
            tail = tail->next; // Geser pointer tail ke node
berikutnya
        }
        baru->next = head; // Atur pointer next node baru ke
head
        head = baru; // Perbarui head menjadi node baru
        tail->next = head; // Hubungkan node terakhir dengan
head
    }
}

// Fungsi untuk menyisipkan node baru di belakang linked list
void insertBelakang(string data) {
    buatNode(data); // Buat node baru dengan data yang
diberikan
    if (isEmpty()) { // Jika linked list kosong
        head = baru; // Atur head menjadi node baru
        tail = head; // Atur tail menjadi head
        baru->next = head; // Atur pointer next node baru ke
head
    } else {
        while (tail->next != head) { // Temukan node terakhir
            tail = tail->next; // Geser pointer tail ke node
berikutnya
        }
        tail->next = baru; // Hubungkan node terakhir dengan
node baru
        baru->next = head; // Atur pointer next node baru ke
head
    }
}

// Fungsi untuk menyisipkan node baru di tengah linked list
pada posisi tertentu
void insertTengah(string data, int posisi) {
    if (isEmpty()) { // Jika linked list kosong
        head = baru; // Atur head menjadi node baru
        tail = head; // Atur tail menjadi head

```

```

        baru->next = head; // Atur pointer next node baru ke
head
    } else {
        baru->data = data; // Atur nilai data pada node baru
        int nomor = 1; // Variabel untuk menyimpan nomor
posisi
        bantu = head; // Pointer untuk menelusuri linked list

        // Temukan posisi node sebelum posisi yang dituju
        while (nomor < posisi - 1) {
            bantu = bantu->next; // Geser pointer bantu
            nomor++; // Tambahkan 1 ke nomor
        }

        baru->next = bantu->next; // Atur pointer next node
baru
        bantu->next = baru; // Hubungkan node sebelumnya
dengan node baru
    }
}

// Fungsi untuk menghapus node dari depan linked list
void hapusDepan() {
    if (!isEmpty()) { // Jika linked list tidak kosong
        hapus = head; // Simpan pointer ke node yang akan
dihapus
        tail = head; // Atur tail menjadi head

        // Jika linked list hanya memiliki satu node
        if (hapus->next == head) {
            head = NULL; // Atur head menjadi NULL
            tail = NULL; // Atur tail menjadi NULL
            delete hapus; // Hapus node yang disimpan
        } else {
            while (tail->next != hapus) { // Temukan node
terakhir
                tail = tail->next; // Geser pointer tail ke
node berikutnya
            }
            head = head->next; // Atur head baru menjadi node
setelahnya
            tail->next = head; // Hubungkan node terakhir
dengan head
        }
    }
}

```

```

        hapus->next = NULL; // Putuskan hubungan node
yang dihapus dengan linked list
        delete hapus; // Hapus node yang disimpan
    }
} else {
    cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
}
}

// Fungsi untuk menghapus node dari belakang linked list
void hapusBelakang() {
    if (!isEmpty()) { // Jika linked list tidak kosong
        hapus = head; // Simpan pointer ke node yang akan
dihapus
        tail = head; // Atur tail menjadi head

        // Jika linked list hanya memiliki satu node
        if (hapus->next == head) {
            head = NULL; // Atur head menjadi NULL
            tail = NULL; // Atur tail menjadi NULL
            delete hapus; // Hapus node yang disimpan
        } else {
            while (hapus->next != head) { // Temukan node
sebelum node terakhir
                hapus = hapus->next; // Geser pointer hapus
ke node berikutnya
            }
            while (tail->next != hapus) { // Temukan node
sebelum node hapus
                tail = tail->next; // Geser pointer tail ke
node berikutnya
            }
            tail->next = head; // Hubungkan node sebelumnya
dengan head
            hapus->next = NULL; // Putuskan hubungan node
yang dihapus dengan linked list
            delete hapus; // Hapus node yang disimpan
        }
    } else {
        cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
    }
}

```



```

}

// Fungsi untuk menghapus node dari tengah linked list pada
posisi tertentu
void hapusTengah(int posisi) {
    if (!isEmpty()) { // Jika linked list tidak kosong
        int nomor = 1; // Variabel untuk menyimpan nomor
posisi
        bantu = head; // Pointer untuk menelusuri linked list

        // Temukan node sebelum node yang akan dihapus
        while (nomor < posisi - 1) {
            bantu = bantu->next; // Geser pointer bantu
            nomor++; // Tambahkan 1 ke nomor
        }

        hapus = bantu->next; // Simpan pointer ke node yang
akan dihapus
        bantu->next = hapus->next; // Hubungkan node
sebelumnya dengan node setelahnya
        delete hapus; // Hapus node yang disimpan
    } else {
        cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
    }
}

// Fungsi untuk menghapus semua node dari linked list
void clearList() {
    if (head != NULL) { // Jika linked list tidak kosong
        hapus = head->next; // Simpan pointer ke node setelah
head
        while (hapus != head) { // Selama masih ada node yang
tersisa
            bantu = hapus->next; // Simpan pointer ke node
setelah node yang akan dihapus
            delete hapus; // Hapus node yang disimpan
            hapus = bantu; // Atur pointer hapus ke node
berikutnya
        }
        delete head; // Hapus node head
        head = NULL; // Atur head menjadi NULL
    }
}

```

```

        cout << "List berhasil terhapus!" << endl; // Tampilkan
pesan berhasil
    }

// Fungsi untuk menampilkan isi linked list
void tampil() {
    if (!isEmpty()) { // Jika linked list tidak kosong
        tail = head; // Atur tail menjadi head
        do {
            cout << tail->data << " "; // Tampilkan nilai
data node
            tail = tail->next; // Geser pointer tail ke node
berikutnya
        } while (tail != head); // Lakukan iterasi sampai
kembali ke head
        cout << endl; // Pindah baris setelah selesai
menampilkan linked list
    } else {
        cout << "List masih kosong!" << endl; // Tampilkan
pesan jika linked list kosong
    }
}

// Fungsi utama
int main() {
    init(); // Inisialisasi linked list
    insertDepan("Ayam"); // Sisipkan node dengan nilai "Ayam"
di depan linked list
    tampil(); // Tampilkan isi linked list
    insertDepan("Bebek"); // Sisipkan node dengan nilai
"Bebek" di depan linked list
    tampil();
    insertBelakang("Cicak"); // Sisipkan node dengan nilai
"Cicak" di belakang linked list
    tampil();
    insertBelakang("Domba"); // Sisipkan node dengan nilai
"Domba" di belakang linked list
    tampil();
    hapusBelakang(); // Hapus node terakhir dari linked list
    tampil();
    hapusDepan(); // Hapus node pertama dari linked list
    tampil();
}

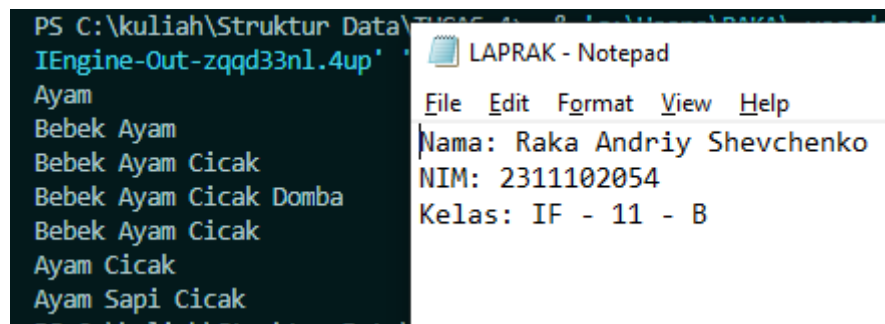
```

```

    insertTengah("Sapi", 2); // Sisipkan node dengan nilai
    "Sapi" di posisi kedua linked list
    tampil();
    hapusTengah(2); // Hapus node pada posisi kedua dari
    linked list
    tampil();
    return 0;
}

```

Output:



The screenshot shows two overlapping windows. On the left is a terminal window with a black background and white text, displaying the output of a C++ program. The output lists the contents of a linked list after several operations: 'Ayam', 'Bebek Ayam', 'Bebek Ayam Cicak', 'Bebek Ayam Cicak Domba', 'Bebek Ayam Cicak', 'Ayam Cicak', and 'Ayam Sapi Cicak'. On the right is a Notepad window titled 'LAPRAK - Notepad'. It has a standard menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains personal information: 'Nama: Raka Andriy Shevchenko', 'NIM: 2311102054', and 'Kelas: IF - 11 - B'.

Deskripsi:

Program C++ di atas mengimplementasikan operasi dasar pada linked list menggunakan struktur data yang mengandung string sebagai data yang disimpan dalam setiap node. Karena penjelasan detail sudah ada di source code, jadi saya akan menjelaskan alur prograamnya. Alur program dimulai dengan inisialisasi linked list, di mana beberapa node disisipkan di depan dan di belakang. Setiap fungsi memastikan penyesuaian pointer yang tepat. Program kemudian menampilkan isi linked list setelah setiap operasi yang dilakukan.

## C. Unguided

### a. Unguided 1

Source code:

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

// Definisi struktur Node untuk Linked List
struct Node {
    string nama;        // Nama mahasiswa yang disimpan dalam
node
    intptr_t nim;       // NIM mahasiswa yang disimpan dalam
node
    Node* next;        // Pointer yang menunjukkan ke node
berikutnya
};

Node* head; // Pointer ke kepala/awal linked list
Node* tail; // Pointer ke ekor/akhir linked list

// Fungsi untuk menginisialisasi linked list
void init() {
    head = nullptr; // Atur pointer head menjadi nullptr
    tail = nullptr; // Atur pointer tail menjadi nullptr
}

// Fungsi untuk memeriksa apakah linked list kosong
bool isEmpty() {
    return head == nullptr; // Mengembalikan true jika head
adalah nullptr
}

// Fungsi untuk menyisipkan node baru di depan linked list
void insertDepan(string nama, intptr_t nim) {
    Node* baru = new Node; // Buat node baru
    baru->nama = nama;      // Atur nama mahasiswa pada node
baru
    baru->nim = nim;        // Atur NIM mahasiswa pada node
baru
    baru->next = nullptr;   // Atur pointer next node baru
menjadi nullptr
}
```

```

        if (isEmpty()) {           // Jika linked list kosong
            head = tail = baru; // Atur head dan tail menjadi node
baru
        } else {
            baru->next = head; // Atur pointer next node baru
menuju head
            head = baru;       // Perbarui head menjadi node baru
        }
    }

// Fungsi untuk menyisipkan node baru di belakang linked list
void insertBelakang(string nama, intptr_t nim) {
    Node* baru = new Node; // Buat node baru
    baru->nama = nama;      // Atur nama mahasiswa pada node
baru
    baru->nim = nim;        // Atur NIM mahasiswa pada node
baru
    baru->next = nullptr; // Atur pointer next node baru
menjadi nullptr

    if (isEmpty()) {           // Jika linked list kosong
        head = tail = baru; // Atur head dan tail menjadi node
baru
    } else {
        tail->next = baru; // Atur pointer next tail menuju
node baru
        tail = baru;      // Perbarui tail menjadi node baru
    }
}

// Fungsi untuk menampilkan data mahasiswa dalam linked list
void tampilkanData() {
    if (isEmpty()) { // Jika linked list kosong
        cout << "Linked list kosong." << endl;
    } else {
        cout << "DATA MAHASISWA" << endl;
        cout << "NAMA\t\t NIM" << endl;
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->nama << "\t" << temp->nim << endl;
            temp = temp->next;
        }
    }
}

```

```

    }
}

// Fungsi untuk menyisipkan node baru di antara dua node yang
// sudah ada
void addBetween(string nama, intptr_t nim, int posisi) {
    if (posisi < 1) {
        cout << "Posisi tidak valid" << endl;
        return;
    }

    Node* baru = new Node; // Buat node baru
    baru->nama = nama;      // Atur nama mahasiswa pada node
    baru          // Atur NIM mahasiswa pada node
    baru->nim = nim;
    baru->next = nullptr;   // Atur pointer next node baru
    menjadi nullptr

    if (isEmpty()) { // Jika Linked list kosong
        head = tail = baru;
    } else if (posisi == 1) { // Jika posisi adalah di depan
        baru->next = head;
        head = baru;
    } else { // Jika posisi di tengah atau belakang
        Node* bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1 && bantu->next != nullptr) {
            bantu = bantu->next;
            nomor++;
        }
        if (nomor == posisi - 1) {
            baru->next = bantu->next;
            bantu->next = baru;
            if (bantu == tail) {
                tail = baru;
            }
        } else {
            cout << "Posisi tidak valid" << endl;
            delete baru;
        }
    }
}
}

```

```

// Fungsi untuk menghapus node dari depan linked list
void hapusDepan() {
    if (isEmpty()) {
        cout << "Linked list kosong." << endl;
    } else {
        cout << "- Hapus Depan\n\n";
        cout << "Data " << head->nama << " berhasil dihapus."
<< endl;
        Node* hapus = head;
        head = head->next;
        delete hapus;
    }
}

// Fungsi untuk menghapus node dari belakang linked list
void hapusBelakang() {
    if (isEmpty()) {
        cout << "Linked list kosong." << endl;
    } else if (head == tail) {
        cout << "- Hapus Belakang\n\n";
        cout << "Data " << head->nama << " berhasil dihapus."
<< endl;
        delete head;
        head = tail = nullptr;
    } else {
        Node* bantu = head;
        while (bantu->next != tail) {
            bantu = bantu->next;
        }
        cout << "- Hapus Belakang\n\n";
        cout << "Data " << tail->nama << " berhasil dihapus."
<< endl;
        delete tail;
        tail = bantu;
        tail->next = nullptr;
    }
}

// Fungsi untuk menghapus node berdasarkan posisi
void hapusTengah(int posisi) {
    if (isEmpty()) {
        cout << "Linked list kosong." << endl;

```

```

    } else if (posisi == 1) {
        hapusDepan();
    } else {
        Node* bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1 && bantu->next != nullptr) {
            bantu = bantu->next;
            nomor++;
        }
        if (nomor == posisi - 1 && bantu->next != nullptr) {
            cout << "\nData " << bantu->next->nama << "
berhasil dihapus." << endl;
            Node* hapus = bantu->next;
            bantu->next = hapus->next;
            delete hapus;
        } else {
            cout << "Posisi tidak valid" << endl;
        }
    }
}

// Fungsi untuk memperbarui data mahasiswa di depan linked
list
void updateDepan(string namaBaru, intptr_t nimBaru) {
    if (isEmpty()) {
        cout << "Linked list kosong." << endl;
    } else {
        cout << "Data " << head->nama << " telah diganti
dengan data " << namaBaru << "." << endl;
        head->nama = namaBaru;
        head->nim = nimBaru;
    }
}

// Fungsi untuk memperbarui data mahasiswa di belakang linked
list
void updateBelakang(string namaBaru, intptr_t nimBaru) {
    if (isEmpty()) {
        cout << "Linked list kosong." << endl;
    } else {
        cout << "Data " << tail->nama << " telah diganti
dengan data " << namaBaru << "." << endl;

```



```

        tail->nama = namaBaru;
        tail->nim = nimBaru;
    }
}

// Fungsi untuk memperbarui data mahasiswa di tengah linked list
void updateTengah(string namaBaru, intptr_t nimBaru, int
posisi) {
    if (isEmpty()) {
        cout << "Linked list kosong." << endl;
    } else if (posisi == 1) {
        head->nama = namaBaru;
        head->nim = nimBaru;
    } else {
        Node* bantu = head;
        int nomor = 1;
        while (nomor < posisi && bantu != nullptr) {
            bantu = bantu->next;
            nomor++;
        }
        if (bantu != nullptr) {
            cout << "Data " << bantu->nama << " telah diganti
dengan data " << namaBaru << "." << endl;
            bantu->nama = namaBaru;
            bantu->nim = nimBaru;
        } else {
            cout << "Posisi tidak valid" << endl;
        }
    }
}

// Fungsi untuk menghapus seluruh data dari linked list
void hapusSeluruhData() {
    Node* current = head;
    Node* next;

    while (current != nullptr) {
        next = current->next; // Simpan pointer ke node
berikutnya
        delete current;      // Hapus node saat ini
        current = next;       // Pindah ke node berikutnya
    }
}

```

```

        // Atur head dan tail menjadi nullptr setelah menghapus
        semua node
        head = tail = nullptr;
    }

// Fungsi utama
int main() {
    init(); // Inisialisasi Linked list

    int pilihan;
    string nama;
    string namaBaruDepan; // Deklarasi di sini
    string namaBaruBelakang; // Deklarasi di sini
    string namaBaruTengah; // Deklarasi di sini
    intptr_t nim;
    intptr_t nimBaruDepan;
    intptr_t nimBaruBelakang;
    intptr_t nimBaruTengah;

    do {
        cout << "\nMenu:";
        cout << "\n1. Tambah Data Mahasiswa (Depan)";
        cout << "\n2. Tambah Data Mahasiswa (Belakang)";
        cout << "\n3. Tambah Data Mahasiswa (Tengah)";
        cout << "\n4. Hapus Data Mahasiswa (Depan)";
        cout << "\n5. Hapus Data Mahasiswa (Belakang)";
        cout << "\n6. Hapus Data Mahasiswa (Tengah)";
        cout << "\n7. Perbarui Data Mahasiswa (Depan)";
        cout << "\n8. Perbarui Data Mahasiswa (Belakang)";
        cout << "\n9. Perbarui Data Mahasiswa (Tengah)";
        cout << "\n10. Tampilkan Data Mahasiswa";
        cout << "\n11. Hapus Seluruh Data";
        cout << "\n0. Keluar";
        cout << "\nPilihan Anda: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "- Tambah Depan\n\n";
                cout << "Masukkan Nama : ";
                cin.ignore(); // Membersihkan buffer sebelum
getline

```

```

        getline(cin, nama);
        cout << "Masukkan NIM : ";
        cin >> nim;
        insertDepan(nama, nim);
        cout << "\nData berhasil ditambahkan.";
        break;
    case 2:
        cout << "- Tambah Belakang\n\n";
        cout << "Masukkan Nama : ";
        cin.ignore(); // Membersihkan buffer sebelum
        getline(cin, nama);
        cout << "Masukkan NIM : ";
        cin >> nim;
        insertBelakang(nama, nim);
        cout << "\nData berhasil ditambahkan.";
        break;
    case 3:
        cout << "- Tambah Tengah\n\n";
        int pos;
        cout << "Masukkan Nama : ";
        cin.ignore(); // Membersihkan buffer sebelum
        getline(cin, nama);
        cout << "Masukkan NIM : ";
        cin >> nim;
        cout << "Masukkan posisi : ";
        cin >> pos;
        addBetween(nama, nim, pos);
        cout << "\nData berhasil ditambahkan.";
        break;
    case 4:
        hapusDepan();
        break;
    case 5:
        hapusBelakang();
        break;
    case 6:
        int posHapus;
        cout << "- Hapus Tengah\n\n";
        cout << "Masukkan posisi : ";
        cin >> posHapus;
        hapusTengah(posHapus);

```

```

        break;
    case 7:
        cout << "- Perbarui Depan\n\n";
        cout << "Masukkan Nama Baru : ";
        cin.ignore(); // Membersihkan buffer sebelum
getline

        getline(cin, namaBaruDepan);
        cout << "Masukkan NIM Baru : ";
        cin >> nimBaruDepan;
        updateDepan(namaBaruDepan, nimBaruDepan);
        break;
    case 8:
        cout << "- Perbarui Belakang\n\n";
        cout << "Masukkan Nama Baru : ";
        cin.ignore(); // Membersihkan buffer sebelum
getline

        getline(cin, namaBaruBelakang);
        cout << "Masukkan NIM Baru : ";
        cin >> nimBaruBelakang;
        updateBelakang(namaBaruBelakang,
nimBaruBelakang);
        break;
    case 9:
        cout << "- Perbarui Tengah\n\n";
        int posUpdate;
        cout << "Masukkan posisi : ";
        cin >> posUpdate;
        cout << "Masukkan Nama Baru : ";
        cin.ignore(); // Membersihkan buffer sebelum
getline

        getline(cin, namaBaruTengah);
        cout << "Masukkan NIM Baru : ";
        cin >> nimBaruTengah;
        updateTengah(namaBaruTengah, nimBaruTengah,
posUpdate);
        break;
    case 10:
        tampilkanData();
        break;
    case 11:
        cout << "Menghapus seluruh data..." << endl;
        hapusSeluruhData();

```

```

        cout << "Seluruh data berhasil dihapus." <<
endl;
        break;
    case 0:
        cout << "Program berakhir." << endl;
        break;
    default:
        cout << "Pilihan tidak valid. Silakan coba
lagi." << endl;
    }
} while (pilihan != 0);

return 0;
}

```

Output:

1.

a. Menu

```

Menu:
1. Tambah Data Mahasiswa (Depan)
2. Tambah Data Mahasiswa (Belakang)
3. Tambah Data Mahasiswa (Tengah)
4. Hapus Data Mahasiswa (Depan)
5. Hapus Data Mahasiswa (Belakang)
6. Hapus Data Mahasiswa (Tengah)
7. Perbarui Data Mahasiswa (Depan)
8. Perbarui Data Mahasiswa (Belakang)
9. Perbarui Data Mahasiswa (Tengah)
10. Tampilkan Data Mahasiswa
11. Hapus Seluruh Data
0. Keluar
Pilihan Anda: 

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

b. Operasi Tambah

```

Pilihan Anda: 1
- Tambah Depan

Masukkan Nama : Raka
Masukkan NIM : 2311102054

Data berhasil ditambahkan.

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

```
- Tambah Belakang  
  
Masukkan Nama : Adam  
Masukkan NIM : 2311102056  
  
Data berhasil ditambahkan.  
Menu
```

```
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

```
- Tambah Tengah  
  
Masukkan Nama : Akmal  
Masukkan NIM : 2311102055  
Masukkan posisi : 2  
  
Data berhasil ditambahkan.
```

```
LAPRAK - Notepad  
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

c. Operasi Hapus

```
- Hapus Depan  
  
Data Rian berhasil dihapus.
```

```
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

```
- Hapus Belakang  
  
Data Ical berhasil dihapus.
```

```
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

```
- Hapus Tengah  
  
Masukkan posisi : 2  
  
Data Arli berhasil dihapus.
```

```
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

d. Operasi Ubah

```
- Perbarui Depan  
  
Masukkan Nama Baru : Rian  
Masukkan NIM Baru : 2311102070  
Data Raka telah diganti dengan data Rian.
```

```
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

```
- Perbarui Belakang  
  
Masukkan Nama Baru : Ical  
Masukkan NIM Baru : 2311102072  
Data Adam telah diganti dengan data Ical.
```

```
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

```

- Perbarui Tengah

Masukkan posisi : 2
Masukkan Nama Baru : Arli
Masukkan NIM Baru : 2311102071
Data Akmal telah diganti dengan data Arli.

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko  
 NIM: 2311102054  
 Kelas: IF - 11 - B

e. Tampil Data

1) Ada data

```

DATA MAHASISWA
NAMA      NIM
Raka      2311102054
Akmal     2311102055
Adam      2311102056

```

File Edit Format View Help

Nama: Raka Andriy Shevchenko  
 NIM: 2311102054  
 Kelas: IF - 11 - B

2) Data kosong

```

Pilihan Anda: 10
Linked list kosong.

```

Nama: Raka Andriy Shevchenko  
 NIM: 2311102054  
 Kelas: IF - 11 - B

2. Data:

```

DATA MAHASISWA
NAMA      NIM
Jawad     23300001
Raka      2311102054
Farrel    23300003
Denis     23300005
Anis      23300008
Bowo      23300015
Gahar     23300040
Udin      23300048
Ucok      23300050
Budi      23300099

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko  
 NIM: 2311102054  
 Kelas: IF - 11 - B

3.

a)

```

- Tambah Tengah

Masukkan Nama : Wati
Masukkan NIM : 23300004
Masukkan posisi : 4

Data berhasil ditambahkan.

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko  
 NIM: 2311102054  
 Kelas: IF - 11 - B

b)

```
- Hapus Tengah  
  
Masukkan posisi : 5  
  
Data Denis berhasil dihapus.
```

```
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

c)

```
- Tambah Depan  
  
Masukkan Nama : Owi  
Masukkan NIM : 23300000  
  
Data berhasil ditambahkan.
```

```
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

d)

```
- Tambah Belakang  
  
Masukkan Nama : David  
Masukkan NIM : 233000100  
  
Data berhasil ditambahkan.
```

```
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

e)

```
- Perbarui Tengah  
  
Masukkan posisi : 9  
Masukkan Nama Baru : Idin  
Masukkan NIM Baru : 23300045  
Data Udin telah diganti dengan data Idin.
```

```
File Edit Format View Help  
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```

f)

```
- Perbarui Belakang  
  
Masukkan Nama Baru : Lucy  
Masukkan NIM Baru : 23300101  
Data David telah diganti dengan data Lucy.
```

```
Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B
```



g)

```
- Hapus Depan
Data Owi berhasil dihapus.
```

```
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

h)

```
- Perbarui Depan
Masukkan Nama Baru : Bagus
Masukkan NIM Baru : 23300002
Data Jawad telah diganti dengan data Bagus.
```

```
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

i)

```
- Hapus Belakang
Data Lucy berhasil dihapus.
```

```
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

j)

```
DATA MAHASISWA
NAMA      NIM
Bagas     23300002
Raka      2311102054
Farrel    23300003
Wati      23300004
Anis      23300008
Bowo      23300015
Gahar     23300040
Idin      23300045
Ucok      23300050
Budi      23300099
```

```
LAPRAK - Notepad
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

Deskripsi:

Program C++ di atas adalah implementasi dari sebuah aplikasi manajemen data mahasiswa menggunakan linked list. Program dimulai dengan inisialisasi linked list dan menampilkan menu pilihan untuk pengguna. Pengguna dapat memilih antara menambahkan data mahasiswa di depan, belakang, atau di tengah linked list, menghapus data dari depan, belakang, atau di tengah linked list berdasarkan posisi tertentu, memperbarui data mahasiswa di depan,

belakang, atau di tengah linked list, menampilkan seluruh data mahasiswa yang tersimpan, dan menghapus seluruh data dari linked list. Setiap fungsi memiliki mekanisme untuk memastikan penyesuaian pointer yang tepat saat operasi-operasi tersebut dilakukan, seperti menambahkan, menghapus, atau memperbarui node. Loop utama dalam program memastikan bahwa menu ditampilkan terus menerus hingga pengguna memilih untuk keluar dari program. Penjelasan lengkap per block dan atau per line sudah ada di kode.

#### D. Kesimpulan

- Linked List adalah sebuah jenis data yang digunakan untuk menyimpan dan mengelola data secara dinamis.
- Linked List dapat dibagi menjadi Linked List Non-Circular dan Linked List Circular.
- Linked List Non-Circular hanya memiliki hubungan dengan simpul berikutnya dalam urutan, sementara Linked List Circular memiliki hubungan yang berlangsung dengan simpul pertama dan terakhir dalam urutan.
- Linked List Non-Circular dapat dibagi menjadi Linked List Singly dan Linked List Doubly, sedangkan Linked List Circular dapat dibagi menjadi Linked List Singly Circular dan Linked List Doubly Circular.
- Linked List Non-Circular dan Linked List Circular memiliki kelebihan dan kelemahan masing-masing, seperti keunggulan dalam implementasi dan kelemahan dalam pengelolaan data.
- Linked List Non-Circular dan Linked List Circular dapat digunakan dalam berbagai aplikasi, seperti sistem operasi, pengembangan permainan, pemutaran musik dan video, pengelolaan buffer, dan cache LRU (Least Recently Used).

## E. Referensi

- [1] Annisa. Pengertian Linked List: Struktur Data dalam Pemrograman. Diakses dari <https://fikti.umsu.ac.id/pengertian-linked-list-struktur-data-dalam-pemrograman/>
- [2] Prepbytes. Everything about Circular Linked List. Diakses dari <https://www.prepbytes.com/blog/linked-list/circular-linked-list-introduction-and-applications/>