LAPORAN PRAKTIKUM

MODUL IV
LINKED LIST CIRCULAR DAN NON CIRCULAR

Disusun oleh:
Syafanida Khakiki
NIM: 2311102005

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

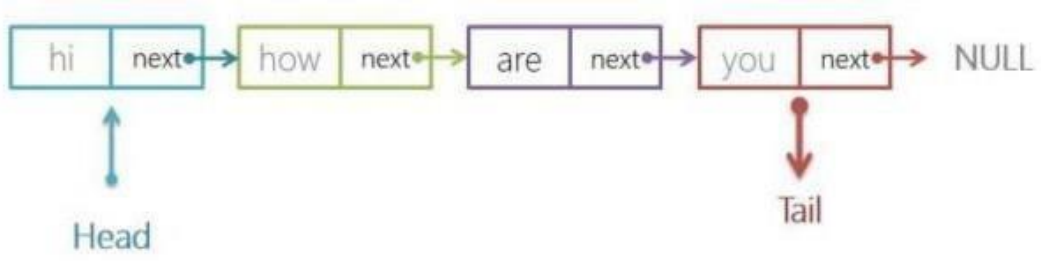
1. Praktikan dapat mengetahui dan memahami linked list circular dan non circular.
2. Praktikan dapat membuat linked list circular dan non circular.
3. Praktikan dapat mengaplikasikan atau menerapkan linked list circular dan non circular pada program yang dibuat.

BAB II

DASAR TEORI

1. Linked List Non Circular

Linked list non circular merupakan *linked list* dengan node pertama (head) dan node terakhir (tail) yang tidak saling terhubung. Pointer terakhir (tail) pada *Linked List* ini selalu bernilai '***NULL***' sebagai pertanda data terakhir dalam *list*-nya. *Linked list non circular* dapat digambarkan sebagai berikut.



Gambar 1 *Single Linked List Non Circular*

OPERASI PADA LINKED LIST NON CIRCULAR

1. Deklarasi Simpul (Node)

```
struct node
{
     int data;
     node *next;
};
```

2. Membuat dan Menginisialisasi Pointer Head dan Tail

```
node *head, *tail;
void init()
{
     head = NULL;
     tail = NULL;
};
```

3. Pengecekan Kondisi Linked List

```
bool isEmpty()
{
if (head == NULL && tail == NULL)
{
return true;
}
else
{
return false;
}
}
```

4. Penambahan Simpul (Node)

```
void insertBelakang(string dataUser)
{
if (isEmpty() == true)
{
node *baru = new node; baru->data =
dataUser; head = baru;
tail = baru;
baru->next = NULL;
}



else
{
node *baru = new node; baru->data =
dataUser; baru->next = NULL; tail->next =
baru; tail = baru;
}
};
```

5. Penghapusan Simpul (Node)

```
void hapusDepan()
{
     if (isEmpty() == true)
     {
          cout << "List kosong!" << endl;
     }
     else
     {
          node *helper;
          helper = head;
          if (head == tail)
          {
               head = NULL; tail =
               NULL; delete
               helper;
          }
          else
               head = head->next;
          helper->next = NULL;
          delete helper;
        }
    }
}
```

6. Tampil Data Linked List

```
void tampil()
{
     if (isEmpty() == true)
     {
          cout << "List kosong!" << endl;
     }
     else
     {
          node *helper;
          helper = head;
          while (helper != NULL)
          {
               cout << helper->data << ends;
               helper = helper->next;
          }
     }
}
```

OPERASI PADA LINKED LIST CIRCULAR

1. Deklarasi Simpul (Node)

```
struct Node
{
     string data;
     Node *next;
};
```

2. Membuat dan Menginisialisasi Pointer Head dan Tail

```
Node *head, *tail, *baru, *bantu, *hapus;

void init()
{
     head = NULL;
     tail = head;
}
```

3. Pengecekan Kondisi Linked List

```
int isEmpty()
{
     if (head == NULL) return
          1; // true
     else
          return 0; // false
}
```

4. Pembuatan Simpul (Node)

```
void buatNode(string data)
{
     baru = new Node;
     baru->data = data;
     baru->next = NULL;
}
```

BAB III
GUIDED

1. **Guided 1**

   Source code :

```
 #include <iostream>
using namespace std;
/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node
struct Node
{
    int data;
    Node *next;
};
Node *head;
Node *tail;

// Inisialisasi Node
void init()
{
    head = NULL;
    tail = NULL;
}

// Pengecekan
bool isEmpty()
{
    if (head == NULL)
        return true;
    else
        return false;
}

// Tambah Depan
void insertDepan(int nilai)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
```

```
    }
    else
    {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(int nilai)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }

    else
    {
        tail->next = baru;
        tail = baru;
    }
}

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

// Tambah Tengah
void insertTengah(int data, int posisi)
{
```

```
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi diluar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else

    {
        Node *baru, *bantu;
        baru = new Node();
        baru->data = data;
        // tranversing
        bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
        }

        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Depan
void hapusDepan()
{
    Node *hapus;
    if (isEmpty() == false)
    {
        if (head->next != NULL)
        {
            hapus = head;
            head = head->next;
            delete hapus;
        }
        else
        {
            head = tail = NULL;
        }
    }
```

```
    else
    {
        cout << "List kosong!" << endl;
    }
}

// Hapus Belakang
void hapusBelakang()
{
    Node *hapus;
    Node *bantu;
    if (isEmpty() == false)
    {
        if (head != tail)
        {
            hapus = tail;
            bantu = head;
            while (bantu->next != tail)
            {
                bantu = bantu->next;
            }
            tail = bantu;
            tail->next = NULL;
            delete hapus;
        }
        else
        {
            head = tail = NULL;
        }
    }

    else
    {
        cout << "List kosong!" << endl;
    }
}

// Hapus Tengah
void hapusTengah(int posisi)
{
    Node *bantu, *hapus, *sebelum;
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi di luar jangkauan" << endl;
    }
```

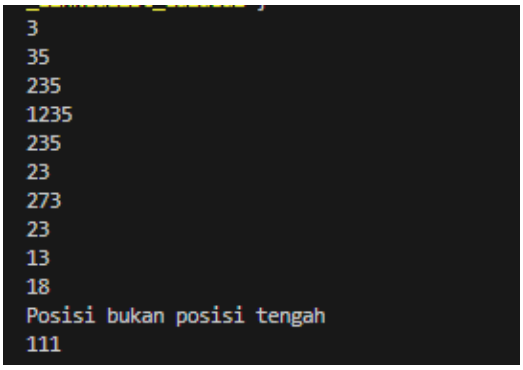
```
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        int nomor = 1;
        bantu = head;
        while (nomor <= posisi)
        {
            if (nomor == posisi - 1)
            {
                sebelum = bantu;
            }
            if (nomor == posisi)
            {
                hapus = bantu;
            }
            bantu = bantu->next;
            nomor++;
        }
        sebelum->next = bantu;
        delete hapus;
    }
}

// Ubah Depan
void ubahDepan(int data)
{
    if (isEmpty() == 0)
    {
        head->data = data;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah Tengah
void ubahTengah(int data, int posisi)
{
    Node *bantu;
    if (isEmpty() == 0)
    {
```

```
        if (posisi < 1 || posisi > hitungList())
        {
            cout << "Posisi di luar jangkauan" << endl;
        }
        else if (posisi == 1)
        {
        }
        else
        {
            cout << "Posisi bukan posisi tengah" << endl;

            bantu = head;
            int nomor = 1;
            while (nomor < posisi)
            {
                bantu = bantu->next;
                nomor++;
            }
            bantu->data = data;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah Belakang
void ubahBelakang(int data)
{
    if (isEmpty() == 0)
    {
        tail->data = data;
    }

    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus List
void clearList()
{
```

```
    Node *bantu, *hapus;
    bantu = head;
    while (bantu != NULL)
    {
        hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = tail = NULL;
    cout << "List berhasil terhapus!" << endl;
}

// Tampilkan List
void tampil()
{
    Node *bantu;
    bantu = head;
    if (isEmpty() == false)
    {
        while (bantu != NULL)
        {
            cout << bantu->data << ends;
            bantu = bantu->next;
        }

        cout << endl;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

int main()
{
    init();
    insertDepan(3);
    tampil();
    insertBelakang(5);
    tampil();
    insertDepan(2);
    tampil();
    insertDepan(1);
    tampil();
    hapusDepan();
```

```
    tampil();
    hapusBelakang();
    tampil();
    insertTengah(7, 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1);
    tampil();
    ubahBelakang(8);
    tampil();
    ubahTengah(11, 2);
    tampil();

 return 0;
}
```

Output :



Deskripsi program :

Program ini dibuat untuk mengimplementasikan struktur data **Single Linked List Non-Circular**.

Fungsi-fungsi:

- init(): Inisialisasi list, membuat list kosong.
- isEmpty(): Mengecek apakah list kosong atau tidak.
- insertDepan(nilai): Menambahkan node baru di depan list.
- insertBelakang(nilai): Menambahkan node baru di belakang list.
- hitungList(): Menghitung jumlah node dalam list.
- insertTengah(data, posisi): Menambahkan node baru di posisi tertentu dalam list.
- hapusDepan(): Menghapus node di depan list.
- hapusBelakang(): Menghapus node di belakang list.
- hapusTengah(posisi): Menghapus node di posisi tertentu dalam list.
- ubahDepan(data): Mengubah data node di depan list.
- ubahTengah(data, posisi): Mengubah data node di posisi tertentu dalam list.

- ubahBelakang(data): Mengubah data node di belakang list.
- clearList(): Menghapus semua node dalam list.
- tampil(): Menampilkan data dalam list.

Penjelasan Fungsi:

- init() : Fungsi ini membuat list kosong dengan menset variabel head dan tail ke NULL.
- isEmpty() : Fungsi ini mengembalikan nilai true jika list kosong, dan false jika list tidak kosong.
- insertDepan(nilai) : Fungsi ini membuat node baru dengan data nilai, kemudian menambahkan node baru di depan list.
    - Jika list kosong, node baru menjadi head dan tail.
    - Jika list tidak kosong, node baru ditambahkan di awal list dan head diubah ke node baru.
- insertBelakang(nilai) : Fungsi ini membuat node baru dengan data nilai, kemudian menambahkan node baru di belakang list.
    - Jika list kosong, node baru menjadi head dan tail.
    - Jika list tidak kosong, node baru ditambahkan di akhir list dan tail diubah ke node baru.
- hitungList() : Fungsi ini menghitung jumlah node dalam list dengan melakukan traversal list dari awal sampai akhir.
- insertTengah(data, posisi) : Fungsi ini membuat node baru dengan data data, kemudian menambahkan node baru di posisi tertentu dalam list.
    - Fungsi ini melakukan traversal list sampai ke posisi yang diberikan.
    - Node baru ditambahkan setelah node pada posisi yang diberikan.
- hapusDepan() : Fungsi ini menghapus node di depan list.
    - Jika list kosong, fungsi ini tidak melakukan apa-apa.
    - Jika list tidak kosong, node di depan list dihapus dan head diubah ke node berikutnya.
- hapusBelakang() : Fungsi ini menghapus node di belakang list.
    - Jika list kosong, fungsi ini tidak melakukan apa-apa.
    - Jika list tidak kosong, node di belakang list dihapus dan tail diubah ke node sebelumnya.
- hapusTengah(posisi) : Fungsi ini menghapus node di posisi tertentu dalam list.
    - Fungsi ini melakukan traversal list sampai ke posisi yang diberikan.
    - Node pada posisi yang diberikan dihapus.
- ubahDepan(data) : Fungsi ini mengubah data node di depan list dengan data.
    - Jika list kosong, fungsi ini tidak melakukan apa-apa.
    - Jika list tidak kosong, data node di depan list diubah dengan data.
- ubahTengah(data, posisi) : Fungsi ini mengubah data node di posisi tertentu dalam list dengan data.
    - Fungsi ini melakukan traversal list sampai ke posisi yang diberikan.
    - Data node pada posisi yang diberikan diubah dengan data.
- ubahBelakang(data) : Fungsi ini mengubah data node di belakang list dengan data.
    - Jika list kosong, fungsi ini tidak melakukan apa-apa.
    - Jika list tidak kosong, data node di belakang list diubah dengan data.
- clearList() : Fungsi ini menghapus semua node dalam list.
    - Fungsi ini melakukan traversal list dan menghapus setiap node satu per satu.
    - Fungsi ini juga mengosongkan variabel head dan tail.
- tampil() : Fungsi ini menampilkan data dalam list.
    - Fungsi ini melakukan traversal list dan mencetak data dari setiap node ke layar.

2. Guided 2

Source Code :

```
      #include <iostream>
using namespace std;
/// PROGRAM SINGLE LINKED LIST CIRCULAR

// Deklarasi Struct Node
struct Node
{
    string data;
    Node *next;
};
Node *head, *tail, *baru, *bantu, *hapus;
void init()
{
    head = NULL;
    tail = head;
}

// Pengecekan
int isEmpty()
{
    if (head == NULL)
        return 1; // true
    else
        return 0; // false
}

// Buat Node Baru
void buatNode(string data)
{
    baru = new Node;
    baru->data = data;
    baru->next = NULL;
}

// Hitung List
int hitungList()
{
    bantu = head;
    int jumlah = 0;
```

```
    while (bantu != NULL)
    {
        jumlah++;
        bantu = bantu->next;
    }

    return jumlah;
}

// Tambah Depan
void insertDepan(string data)
{
    // Buat Node baru
    buatNode(data);

    if (isEmpty() == 1)
    {
        head = baru;
        tail = head;
        baru->next = head;
    }
    else
    {
        while (tail->next != head)
        {
            tail = tail->next;
        }
        baru->next = head;
        head = baru;
        tail->next = head;
    }
}

// Tambah Belakang
void insertBelakang(string data)
{
    // Buat Node baru
    buatNode(data);

    if (isEmpty() == 1)
    {
        head = baru;
        tail = head;
        baru->next = head;
    }
```

```
    else
    {
        while (tail->next != head)
        {
            tail = tail->next;
        }

        tail->next = baru;
        baru->next = head;
    }
}

// Tambah Tengah
void insertTengah(string data, int posisi)
{
    if (isEmpty() == 1)
    {
        head = baru;
        tail = head;
        baru->next = head;
    }
    else
    {
        baru->data = data;
        // transversing
        int nomor = 1;
        bantu = head;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Depan
void hapusDepan()
{
    if (isEmpty() == 0)
    {
        hapus = head;
        tail = head;
```

```
        if (hapus->next == head)
        {
            head = NULL;
            tail = NULL;
        }
        else
        {
            delete hapus;

            while (tail->next != hapus)
            {
                tail = tail->next;
            }
            head = head->next;
            tail->next = head;
            hapus->next = NULL;

            delete hapus;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
// Hapus Belakang
void hapusBelakang()
{
    if (isEmpty() == 0)
    {
        hapus = head;
        tail = head;
        if (hapus->next == head)
        {
            head = NULL;
            tail = NULL;
        }
        else
        {
            delete hapus;

            while (hapus->next != head)
            {
                hapus = hapus->next;
            }
```

```
            while (tail->next != hapus)
            {
                tail = tail->next;
            }
            tail->next = head;
            hapus->next = NULL;

            delete hapus;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus Tengah
void hapusTengah(int posisi)
{
    if (isEmpty() == 0)
    {
        // transversing
        int nomor = 1;
        bantu = head;

        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        hapus = bantu->next;
        bantu->next = hapus->next;

        delete hapus;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus List
void clearList()
{
    if (head != NULL)
```

```
    {
        hapus = head->next;

        while (hapus != head)
        {
            bantu = hapus->next;
            delete hapus;
            hapus = bantu;
        }
        delete head;
        head = NULL;
    }
    cout << "List berhasil terhapus!" << endl;
}

// Tampilkan List
void tampil()
{
    if (isEmpty() == 0)
    {
        tail = head;
        do
        {
            cout << tail->data << ends;
            tail = tail->next;
        } while (tail != head);
        cout << endl;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
int main()
{
    init();
    insertDepan("Ayam");
    tampil();
    insertDepan("Bebek");
    tampil();
    insertBelakang("Cicak");
    tampil();
    insertBelakang("Domba");
    tampil();
    hapusBelakang();
```

```
    tampil();
    hapusDepan();
    tampil();
    insertTengah("Sapi", 2);
    tampil();
    hapusTengah(2);
    tampil();

 return 0;
}
```

Output :

```
Ayam
BebekAyam
BebekAyamCicak
BebekAyamCicakDomba
```

Deskripsi Program :

Program ini dibuat untuk mengimplementasikan struktur data **Single Linked List Circular**.

Fungsi-fungsi:

- init(): Inisialisasi list, membuat list kosong.
- isEmpty(): Mengecek apakah list kosong atau tidak.
- buatNode(data): Membuat node baru dengan data yang diberikan.
- hitungList(): Menghitung jumlah node dalam list.
- insertDepan(data): Menambahkan node baru di depan list.
- insertBelakang(data): Menambahkan node baru di belakang list.
- insertTengah(data, posisi): Menambahkan node baru di posisi tertentu dalam list.
- hapusDepan(): Menghapus node di depan list.
- hapusBelakang(): Menghapus node di belakang list.
- hapusTengah(posisi): Menghapus node di posisi tertentu dalam list.
- clearList(): Menghapus semua node dalam list.
- tampil(): Menampilkan data dalam list.

Penjelasan Fungsi:

- init() : Fungsi ini membuat list kosong dengan menset variabel head dan tail ke NULL.
- isEmpty() : Fungsi ini mengembalikan nilai 1 (true) jika list kosong, dan 0 (false) jika list tidak kosong.
- buatNode(data) : Fungsi ini mengalokasikan memori baru untuk node baru dan mengisi data node dengan data yang diberikan.
- hitungList() : Fungsi ini menghitung jumlah node dalam list dengan melakukan traversal list dari awal sampai akhir.
- insertDepan(data) : Fungsi ini membuat node baru dengan buatNode(data), kemudian menambahkan node baru di depan list.
    - Jika list kosong, node baru menjadi head dan tail.
    - Jika list tidak kosong, node baru ditambahkan di awal list dan head diubah ke node baru.
- insertBelakang(data) : Fungsi ini membuat node baru dengan buatNode(data), kemudian menambahkan node baru di belakang list.
    - Jika list kosong, node baru menjadi head dan tail.
    - Jika list tidak kosong, node baru ditambahkan di akhir list dan tail diubah ke node baru.
- insertTengah(data, posisi) : Fungsi ini membuat node baru dengan buatNode(data), kemudian menambahkan node baru di posisi tertentu dalam list.
    - Fungsi ini melakukan traversal list sampai ke posisi yang diberikan.
    - Node baru ditambahkan setelah node pada posisi yang diberikan.
- hapusDepan() : Fungsi ini menghapus node di depan list.
    - Jika list kosong, fungsi ini tidak melakukan apa-apa.
    - Jika list tidak kosong, node di depan list dihapus dan head diubah ke node berikutnya.
- hapusBelakang() : Fungsi ini menghapus node di belakang list.
    - Jika list kosong, fungsi ini tidak melakukan apa-apa.
    - Jika list tidak kosong, node di belakang list dihapus dan tail diubah ke node sebelumnya.
- hapusTengah(posisi) : Fungsi ini menghapus node di posisi tertentu dalam list.
    - Fungsi ini melakukan traversal list sampai ke posisi yang diberikan.
    - Node pada posisi yang diberikan dihapus.
- clearList() : Fungsi ini menghapus semua node dalam list.
    - Fungsi ini melakukan traversal list dan menghapus setiap node satu per satu.
    - Fungsi ini juga mengosongkan variabel head dan tail.
- tampil() : Fungsi ini menampilkan data dalam list.
    - Fungsi ini melakukan traversal list dan mencetak data dari setiap node ke layar.

BAB IV
UNGUIDED

Unguided

Source code :

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

struct Node
{
    string nama;
    string nim;
    Node *next;
};

bool isEmpty(Node *head)
{
    return head == NULL;
}

Node* buatNode(string nama, string nim)
{
    Node *baru = new Node;
    baru->nama = nama;
    baru->nim = nim;
    baru->next = NULL;
    return baru;
}

Node* tambahDepan(Node *head, string nama, string nim)
{
    Node *baru = buatNode(nama, nim);
    if (isEmpty(head))
    {
        return baru;
    }
    baru->next = head;
    return baru;
}
```

```
Node* tambahBelakang(Node *head, string nama, string nim)
{
    Node *baru = buatNode(nama, nim);
    if (isEmpty(head))
    {
        return baru;
    }
    Node *tail = head;
    while (tail->next != NULL)
    {
        tail = tail->next;
    }
    tail->next = baru;
    return head;
}

Node* tambahTengah(Node *head, string nama, string nim, int posisi)
{
    if (posisi < 1)
    {
        cout << "Posisi tidak valid" << endl;
        return head;
    }
    if (posisi == 1)
    {
        cout << "Gunakan tambahDepan untuk menambahkan pada posisi pertama" <<
endl;
        return tambahDepan(head, nama, nim);
    }
    Node *baru = buatNode(nama, nim);
    Node *bantu = head;
    for (int i = 1; i < posisi - 1 && bantu != NULL; i++)
    {
        bantu = bantu->next;
    }
    if (bantu == NULL)
    {
        cout << "Posisi diluar jangkauan" << endl;
        return head;
    }
    baru->next = bantu->next;
    bantu->next = baru;
    return head;
}
```

```
void ubahDepan(Node *head, string nama, string nim)
{
    if (!isEmpty(head))
    {
        head->nama = nama;
        head->nim = nim;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

void ubahBelakang(Node *head, string nama, string nim)
{
    if (!isEmpty(head))
    {
        Node *tail = head;
        while (tail->next != NULL)
        {
            tail = tail->next;
        }
        tail->nama = nama;
        tail->nim = nim;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

void ubahTengah(Node *head, string nama, string nim, int posisi)
{
    if (!isEmpty(head))
    {
        if (posisi < 1)
        {
            cout << "Posisi tidak valid" << endl;
            return;
        }
        Node *bantu = head;
        for (int i = 1; i < posisi && bantu != NULL; i++)
        {
            bantu = bantu->next;
```

```
        }
        if (bantu == NULL)
        {
            cout << "Posisi diluar jangkauan" << endl;
            return;
        }
        bantu->nama = nama;
        bantu->nim = nim;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

Node* hapusDepan(Node *head)
{
    if (!isEmpty(head))
    {
        Node *hapus = head;
        head = head->next;
        delete hapus;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
    return head;
}

Node* hapusBelakang(Node *head)
{
    if (!isEmpty(head))
    {
        Node *hapus = NULL;
        if (head->next == NULL)
        {
            delete head;
            return NULL;
        }
        Node *tail = head;
        while (tail->next->next != NULL)
        {
            tail = tail->next;
        }
```

```
        hapus = tail->next;
        tail->next = NULL;
        delete hapus;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
    return head;
}

Node* hapusTengah(Node *head, int posisi)
{
    if (!isEmpty(head))
    {
        if (posisi < 1)
        {
            cout << "Posisi tidak valid" << endl;
            return head;
        }
        if (posisi == 1)
        {
            return hapusDepan(head);
        }
        Node *bantu = head;
        for (int i = 1; i < posisi - 1 && bantu != NULL; i++)
        {
            bantu = bantu->next;
        }
        if (bantu == NULL || bantu->next == NULL)
        {
            cout << "Posisi diluar jangkauan" << endl;
            return head;
        }
        Node *hapus = bantu->next;
        bantu->next = hapus->next;
        delete hapus;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
    return head;
}
```

```
void hapusList(Node *&head)
{
    while (!isEmpty(head))
    {
        head = hapusDepan(head);
    }
    cout << "List berhasil terhapus!" << endl;
}

int hitungList(Node *head)
{
    int jumlah = 0;
    Node *bantu = head;
    while (bantu != NULL)
    {
        jumlah++;
        bantu = bantu->next;
    }
    return jumlah;
}

void tampil(Node *head)
{
    if (!isEmpty(head))
    {
        Node *bantu = head;
        cout << "=========================" << endl;
        cout << "   DATA MAHASISWA" << endl;
        cout << "=========================" << endl;
        cout << "|   NAMA          |  NIM  |" << endl;
        cout << "--------------------------" << endl;
        while (bantu != NULL)
        {
            cout << "|   " << bantu->nama << "      |  " << bantu->nim <<
"   |" << endl;
            bantu = bantu->next;
        }
        cout << "--------------------------" << endl;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
```

```
int main()
{
    Node *head = NULL;
    int choice, posisi;
    string nama, nim;
    do
    {
        cout << "PROGRAM SINGLE LINKED LIST NON-CIRCULAR" << endl;
        cout << "1. Tambah Depan" << endl;
        cout << "2. Tambah Belakang" << endl;
        cout << "3. Tambah Tengah" << endl;
        cout << "4. Ubah Depan" << endl;
        cout << "5. Ubah Belakang" << endl;
        cout << "6. Ubah Tengah" << endl;
        cout << "7. Hapus Depan" << endl;
        cout << "8. Hapus Belakang" << endl;
        cout << "9. Hapus Tengah" << endl;
        cout << "10. Hapus List" << endl;
        cout << "11. TAMPILKAN" << endl;
        cout << "0. KELUAR" << endl;
        cout << "Pilih Operasi: ";
        cin >> choice;

        switch (choice)
        {
            case 1:
                cout << "-Tambah Depan" << endl;
                cout << "Masukkan Nama : ";
                cin >> nama;
                cout << "Masukkan NIM : ";
                cin >> nim;
                head = tambahDepan(head, nama, nim);
                cout << "Data telah ditambahkan" << endl;
                break;
            case 2:
                cout << "-Tambah Belakang" << endl;
                cout << "Masukkan Nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                cin >> nim;
                head = tambahBelakang(head, nama, nim);
                cout << "Data telah ditambahkan" << endl;
                break;
            case 3:
                cout << "-Tambah Tengah" << endl;
```

```
                cout << "Masukkan Nama : ";
                cin >> nama;
                cout << "Masukkan NIM : ";
                cin >> nim;
                cout << "Masukkan Posisi : ";
                cin >> posisi;
                head = tambahTengah(head, nama, nim, posisi);
                cout << "Data telah ditambahkan" << endl;
                break;
            case 4:
                cout << "Masukkan Nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                cin >> nim;
                ubahDepan(head, nama, nim);
                break;
            case 5:
                cout << "-Ubah Belakang" << endl;
                cout << "Masukkan nama : ";
                cin >> nama;
                cout << "Masukkan NIM : ";
                cin >> nim;
                ubahBelakang(head, nama, nim);
                cout << "Data (nama lama) telah diganti dengan data (nama
baru)" << endl;
                break;
            case 6:
                cout << "Masukkan Nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                cin >> nim;
                cout << "Masukkan posisi: ";
                cin >> posisi;
                ubahTengah(head, nama, nim, posisi);
                break;
            case 7:
                head = hapusDepan(head);
                break;
            case 8:
                cout << "-Hapus Belakang" << endl;
                head = hapusBelakang(head);
                cout << "Data (nama mahasiswa yang dihapus) berhasil dihapus"
<< endl;
                break;
            case 9:
```

```
                cout << "-Hapus Tengah" << endl;
                cout << "Masukkan posisi : ";
                cin >> posisi;
                head = hapusTengah(head, posisi);
                cout << "Data (nama mahasiswa yang dihapus) berhasil dihapus"
<< endl;
                break;
            case 10:
                hapusList(head);
                break;
            case 11:
                tampil(head);
                break;
            case 0:
                cout << "Terima kasih!" << endl;
                break;
            default:
                cout << "Pilihan tidak valid!" << endl;
                break;
        }
    } while (choice != 0);

    return 0;
}
```

1. Buatlah menu untuk menambahkan, mengubah, menghapus, dan melihat Nama dan NIM mahasiswa

Output :

- Tampilan Menu

```
PROGRAM SINGLE LINKED LIST NON-CIRCULAR
1. Tambah Depan
2. Tambah Belakang
3. Tambah Tengah
4. Ubah Depan
5. Ubah Belakang
6. Ubah Tengah
7. Hapus Depan
8. Hapus Belakang
9. Hapus Tengah
10. Hapus List
11. TAMPILKAN
0. KELUAR
Pilih Operasi:
```

- Tampilan Operasi Tambah:

```
-Tambah Depan
Masukkan Nama : Syafa
Masukkan NIM : 2311102005
Data telah ditambahkan
```

```
-Tambah Tengah
Masukkan Nama : Nida
Masukkan NIM : 2311102006
Masukkan Posisi : 2
Data telah ditambahkan
```

```
-Tambah Belakang
Masukkan Nama: Khakiki
Masukkan NIM: 2311102007
Data telah ditambahkan
```

- Tampilan Operasi Ubah:

```
Pilih Operasi: 4
Masukkan Nama: Rofiq
Masukkan NIM: 2311102001
```

```
Pilih Operasi: 6
Masukkan Nama: Nur
Masukkan NIM: 2311102002
Masukkan posisi: 2
```

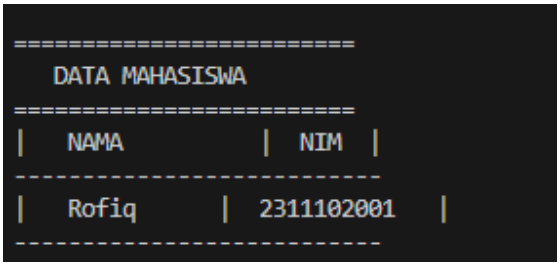
```
-Ubah Belakang
Masukkan nama : Hidayah
Masukkan NIM : 2311102003
Data (nama lama) telah diganti dengan data (nama baru)
```

- Tampilan Operasi Hapus:

```
-Hapus Tengah
Masukkan posisi : 2
Data (nama mahasiswa yang dihapus) berhasil dihapus
```

```
-Hapus Belakang
Data (nama mahasiswa yang dihapus) berhasil dihapus
```

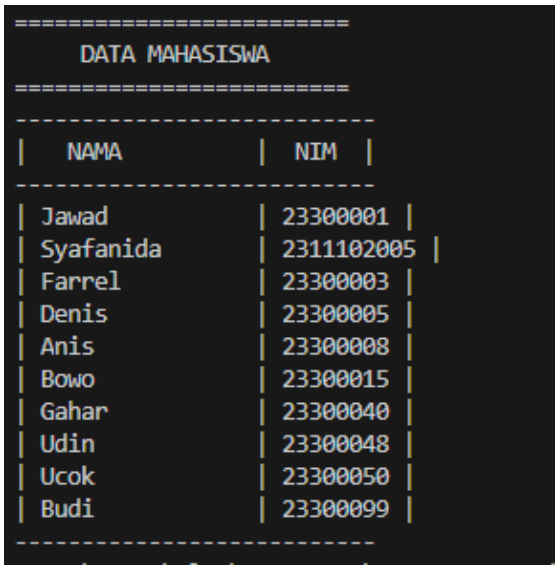
- Tampilan Operasi Tampil:



2. Setelah membuat menu tersebut, masukkan data sesuai urutan berikut, lalu tampilkan data yang telah dimasukkan. (Gunakan insert depan, belakang atau tengah)

Nama	NIM
Jawad	**23300001**
[Nama Anda]	**[NIM Anda]**
Farrel	**23300003**
Denis	**23300005**
Anis	**23300008**
Bowo	**23300015**
Gahar	**23300040**
Udin	**23300048**
Ucok	**23300050**
Budi	**23300099**

Output :



3. Lakukan perintah berikut:

a) Tambahkan data berikut diantara Farrel dan Denis:

Wati **2330004**

```
Pilih Operasi: 3
-Tambah Tengah
Masukkan Nama : Wati
Masukkan NIM : 2330004
Masukkan Posisi : 4
Data telah ditambahkan
```

b) Hapus data Denis

```
Pilih Operasi: 9
-Hapus Tengah
Masukkan posisi : 5
Data (nama mahasiswa yang dihapus) berhasil dihapus
```

c) Tambahkan data berikut di awal:

Owi **2330000**

```
Pilih Operasi: 1
-Tambah Depan
Masukkan Nama : Owi
Masukkan NIM : 2330000
Data telah ditambahkan
```

d) Tambahkan data berikut di akhir:

David **23300100**

```
Pilih Operasi: 2
-Tambah Belakang
Masukkan Nama: David
Masukkan NIM: 23300100
Data telah ditambahkan
```

e) Ubah data Udin menjadi data berikut:

Idin **23300045**

```
Pilih Operasi: 6
Masukkan Nama: Idin
Masukkan NIM: 23300045
Masukkan posisi: 9
```

f) Ubah data terkahir menjadi berikut:

Lucy 23300101

```
Pilih Operasi: 5
-Ubah Belakang
Masukkan nama : Lucy
Masukkan NIM : 23300101
Data (nama lama) telah diganti dengan data (nama baru)
```

g) Hapus data awal

```
Pilih Operasi: 7
```

h) Ubah data awal menjadi berikut:

Bagas 2330002

```
Pilih Operasi: 4
Masukkan Nama: Bagas
Masukkan NIM: 2330002
```

i) Hapus data akhir

```
Pilih Operasi: 8
-Hapus Belakang
Data (nama mahasiswa yang dihapus) berhasil dihapus
```

j) Tampilkan seluruh data

```
==========================
--------------------------
|   NAMA         |  NIM  |
--------------------------
| Bagas          |  2330002 |
| Syafanida      | 2311102005 |
| Farrel         | 23300003 |
| Wati           |  2330004 |
| Anis           | 23300008 |
| Bowo           | 23300015 |
| Gahar          | 23300040 |
| Idin           | 23300045 |
| Ucok           | 23300050 |
| Budi           | 23300099 |
--------------------------
```

Deskripsi Program :

Program ini dibuat untuk mengelola data mahasiswa dalam sebuah struktur data **Single Linked List Non-Circular**.
Berikut adalah beberapa kegunaan dan fungsi dari program ini:

Operasi Dasar:
Menambah Data:

- Tambah Depan: Menambahkan data mahasiswa di awal list.
- Tambah Belakang: Menambahkan data mahasiswa di akhir list.
- Tambah Tengah: Menambahkan data mahasiswa di posisi tertentu dalam list.

- **Mengubah Data:**
  - Ubah Depan: Mengubah data mahasiswa di awal list.
  - Ubah Belakang: Mengubah data mahasiswa di akhir list.
  - Ubah Tengah: Mengubah data mahasiswa di posisi tertentu dalam list.
- **Menghapus Data:**
  - Hapus Depan: Menghapus data mahasiswa di awal list.
  - Hapus Belakang: Menghapus data mahasiswa di akhir list.
  - Hapus Tengah: Menghapus data mahasiswa di posisi tertentu dalam list.

Operasi Lain:

- Hapus List: Menghapus seluruh data dalam list.
- Tampilkan: Menampilkan seluruh data mahasiswa dalam list.

Penjelasan Struktur Program:

Struktur Node**:**

- nama: Menyimpan nama mahasiswa.
- nim: Menyimpan NIM mahasiswa.
- next: Menunjuk ke node berikutnya dalam list.

- **Fungsi-Fungsi:**
  - isEmpty: Mengecek apakah list kosong.
  - buatNode: Membuat node baru dengan data yang diberikan.
  - tambahDepan: Menambahkan node baru di awal list.
  - tambahBelakang: Menambahkan node baru di akhir list.
  - tambahTengah: Menambahkan node baru di posisi tertentu dalam list.
  - ubahDepan: Mengubah data node di awal list.
  - ubahBelakang: Mengubah data node di akhir list.
  - ubahTengah: Mengubah data node di posisi tertentu dalam list.
  - hapusDepan: Menghapus node di awal list.
  - hapusBelakang: Menghapus node di akhir list.

- hapusTengah: Menghapus node di posisi tertentu dalam list.
- hapusList: Menghapus seluruh node dalam list.
- hitungList: Menghitung jumlah node dalam list.
- tampil: Menampilkan seluruh data node dalam list.

Alur Kerja Program:

1. Program dimulai dengan mendeklarasikan struktur Node dan beberapa fungsi untuk mengelola list.
2. Variabel head digunakan untuk menunjuk ke node pertama dalam list.
3. Looping do-while digunakan untuk menjalankan program berulang kali sampai pengguna memilih untuk keluar.
4. Di dalam looping, pengguna dihadapkan dengan menu pilihan untuk melakukan operasi terhadap list.
5. Berdasarkan pilihan pengguna, fungsi-fungsi yang sesuai dijalankan untuk melakukan operasi yang diinginkan.

BAB IV

KESIMPULAN

1. Linked List Non Circular

Definisi:
Linked List Non Circular adalah struktur data linear yang terdiri dari sejumlah node yang dihubungkan satu sama lain melalui pointer. Pada Linked List Non Circular, node pertama (head) dan node terakhir (tail) tidak saling terhubung. Pointer terakhir (tail) selalu menunjuk ke NULL sebagai penanda akhir dari linked list.

Karakteristik:
- Tidak memiliki sambungan dari node terakhir kembali ke node pertama.
- Memiliki head sebagai pointer awal dan tail sebagai pointer akhir.
- Data tersimpan secara berurutan dari head ke tail.

Operasi yang Dapat Dilakukan:
1) Deklarasi Simpul (Node)
2) Membuat dan Menginisialisasi Pointer Head dan Tail
3) Pengecekan Kondisi Linked List
4) Penambahan Simpul (Node) di berbagai posisi
5) Penghapusan Simpul (Node) di berbagai posisi
6) Menampilkan Data Linked List

2. Linked List Circular

Definisi:
Linked List Circular adalah struktur data linear yang terdiri dari sejumlah node yang dihubungkan satu sama lain melalui pointer. Pada Linked List Circular, node terakhir menghubungkan kembali ke node pertama, membentuk lingkaran.

Karakteristik:
- Memiliki sambungan dari node terakhir kembali ke node pertama.
- Memiliki head sebagai pointer awal dan tail sebagai pointer akhir.
- Data tersimpan secara berurutan dari head ke tail, membentuk lingkaran.

Operasi yang Dapat Dilakukan:

1) Deklarasi Simpul (Node)
2) Membuat dan Menginisialisasi Pointer Head dan Tail
3) Pengecekan Kondisi Linked List
4) Penambahan Simpul (Node) di berbagai posisi
5) Penghapusan Simpul (Node) di berbagai posisi
6) Menampilkan Data Linked List

Perbandingan Kedua Jenis Linked List

- Linked List Circular lebih tahan terhadap manipulasi data karena tidak memiliki node yang berdiri sendiri tanpa pointer yang mengarahkan kepadanya.

- Linked List Circular dapat digunakan dalam kasus di mana data perlu diakses berulang kali dalam siklus tertentu, misalnya dalam algoritma penjadwalan.

- **Penggunaan Memori:** Linked List Circular mungkin membutuhkan sedikit lebih banyak penggunaan memori karena pointer tail harus selalu diperbarui untuk mengakomodasi tambahan node.

  Kedua jenis linked list memiliki kegunaan dan aplikasi khususnya masing-masing tergantung pada kebutuhan dari permasalahan yang dihadapi. Pemilihan jenis linked list yang tepat akan mempengaruhi efisiensi dan kemudahan dalam implementasi algoritma yang bersangkutan.

DAFTAR PUSTAKA

Asisten Praktikum (2024). Linked List Circular dan Non Circular. Learning Management System.

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts,problems, Interview Questions*. CareerMonk Publications.