Modul Praktikum

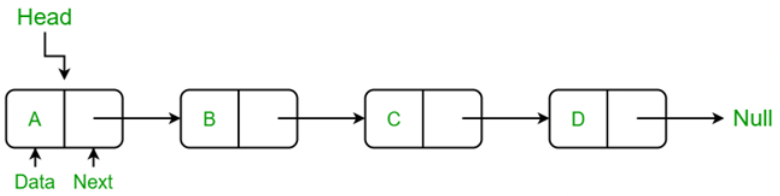
SDAA

3

LINKED LIST

SINGLE LINKED LIST

Linked list merupakan salah satu jenis struktur data yang terdiri dari kumpulan data yang tersusun secara berurutan, saling terhubung, bersifat dinamis, dan terbatas. Setiap elemen di dalam linked list, disebut simpul (node), saling terkait dalam urutan tertentu. Simpul ini biasanya didefinisikan sebagai sebuah struktur atau class yang menyimpan data. Dalam konsepnya, linked list terdiri dari sejumlah node yang dihubungkan secara linear menggunakan pointer.



Gambar 1. Single Linked List

DEKLARASI NODE

Pada linked list, setiap simpul (node) biasanya terdiri dari dua bagian utama: bagian pertama menyimpan data, dan bagian kedua menyimpan referensi atau pointer yang menghubungkan simpul tersebut ke simpul berikutnya. Di dalam bahasa pemrograman seperti C++, simpul ini dapat dideklarasikan menggunakan struktur (struct) atau class. Contoh deklarasi node dalam C++ menggunakan struct.

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

// Membuat node menggunakan struct
struct Node {
    int data;
    Node *next;
};

int main() {
    // Pointer untuk menyimpan alamat node pertama
    Node *HEAD = nullptr;

    return 0;
}
```

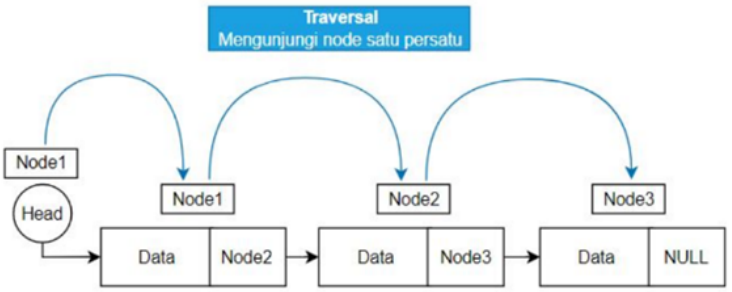
Gambar 2. Deklarasi Node

Setiap node akan berbentuk struct dan memiliki satu buah field bertipe struct yang sama dan yang berfungsi sebagai pointer ke node berikutnya.

- `struct Node` : Kita membuat node menggunakan struct, yaitu keyword yang berfungsi untuk mengelompokan kumpulan variabel ke dalam 1 nama. Di atas kita mengelompokan variabelnya kedalam struct yang bernama Node.
- `Node *next` : artinya kita membuat variabel pointer yang bisa menyimpan alamat yang bertipe data struct Node.
- `Node *HEAD` :Berfungsi untuk menyimpan alamat dari Node pertama, saat deklarasi kita berikan nilai NULL/nullptr/0 karena linked list dianggap masih kosong.

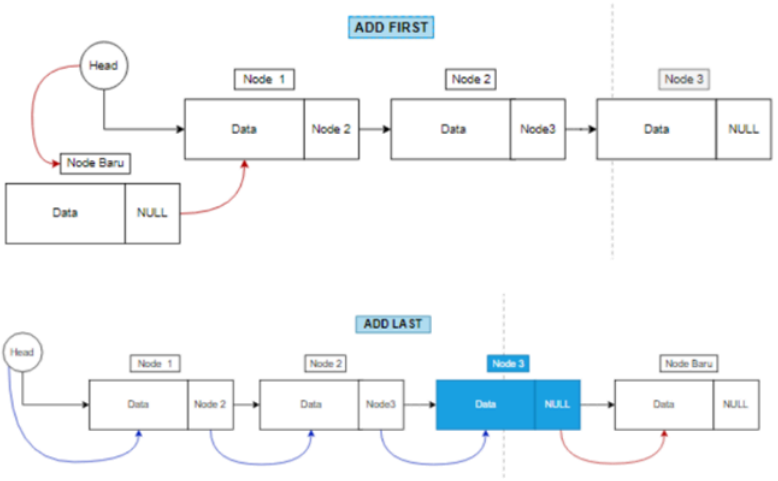
TRAVERSAL

Traversal adalah proses mengunjungi setiap elemen Node secara berurutan, dimulai dari node pertama. Karena linked list tidak memiliki fitur indexing data, kita harus melakukan traversal untuk mencapai Node yang ingin diakses.



Gambar 3. Traversal

MENAMBAH DATA



Gambar 4. Menambah Data di Awal dan Akhir Node

```
void addFirst(Node *&head, int databaru) {
    Node *nodeBaru = new Node;
    nodeBaru->data = databaru;
    nodeBaru->next = head;
    head = nodeBaru;
}

void addLast(Node *&head, int databaru) {
    Node *nodeBaru = new Node;
    nodeBaru->data = databaru;
    nodeBaru->next = nullptr;
    // jika linked list kosong
    if (head == nullptr){
        head = nodeBaru;
        return;
    }
    // jika tidak, traversal ke node terakhir
    Node *temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = nodeBaru;
}
```

MENAMPILKAN DATA

Berikut merupakan source code untuk menampilkan data pada single linked list.

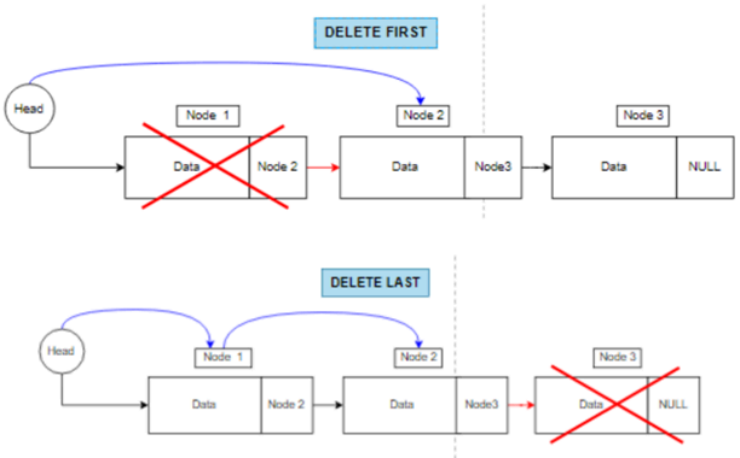
```
void printList(Node *head) {
    if (head == nullptr){
        cout << ">> LinkedList masih kosong <<" << endl;
        return;
    }

    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
```

MENGUBAH DATA

Berikut merupakan source code untuk mengubah data pada single linked list.

```
void updateData(Node *&head) {
    if (head == nullptr){
        cout << "LinkedList masih kosong" << endl;
        return;
    }

    int ubah;
    cout << "Masukan data yang akan diubah : ";
    cin >> ubah;

    Node *temp = head;
    while (temp != nullptr){
        if (temp->data == ubah){
            cout << "Masukan data yang baru : ";
            cin >> temp->data;
            cout << "Data berhasil diubah" << endl;
            return;
        }
        temp = temp->next;
    }
    cout << "Data tidak ditemukan" << endl;
}
```

MENGHAPUS DATA



Gambar 5. Menghapus Data

```
void deleteFirst(Node *&head) {
    if (head == nullptr){
        cout << ">> LinkedList masih kosong <<" << endl;
        return;
    }
    Node *temp = head;
    head = head->next;
    delete temp;
}

void deleteLast(Node *&head) {
    if (head == nullptr){
        cout << ">> LinkedList masih kosong <<" << endl;
        return;
    }

    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }

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

OPERASI PADA LINKED LIST

Operasi pada Linked List mencakup berbagai manipulasi dasar yang dapat dilakukan pada struktur data ini, seperti menambah, menghapus, dan mencari elemen. Tidak seperti array, Linked List memungkinkan pengelolaan data yang lebih fleksibel karena elemen-elemen tidak perlu disimpan secara berurutan dalam memori. Operasi-operasi ini biasanya melibatkan penyesuaian referensi atau "pointer" yang menghubungkan node-node di dalam Linked List.

- Find First
  Fungsi ini mencari elemen pertama dari linked list menggunakan variabel head.

- Find Next
  Fungsi ini mencari elemen sesudah elemen yang ditunjuk saat ini dengan memanggil pengait next yang ada di dalam struct.

- Find Last

  Fungsi ini mencari elemen terakhir dari linked list menggunakan variabel tail.

- isEmpty

  Fungsi ini menentukan apakah linked list kosong atau tidak.

```
int isEmpty(Node *&head){
    if(head == nullptr) {
        return 1;
    } else {
        return 0;
    }
}
```

- Insert

  Istilah Insert berarti menambahkan sebuah simpul baru ke dalam suatu linked list.

- Retrieve

  Fungsi ini mengambil elemen yang ditunjuk. Elemen tersebut lalu dikembalikan oleh fungsi.

- Update

  Fungsi ini mengubah isi dari elemen yang ditunjuk.

- Delete Head

  Fungsi ini menghapus elemen yang ditunjuk head. Head berpindah ke elemen sesudahnya.

- Delete Tail

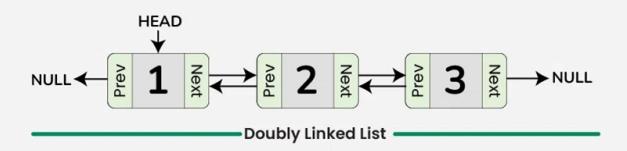
  Fungsi ini menghapus elemen yang ditunjuk tail. Tail berpindah ke elemen sebelumnya.

- Clear

  Fungsi ini menghapus linked list yang sudah ada.

DOUBLE LINKED LIST

Double Linked List adalah elemen-elemen yang dihubungkan dengan dua pointer dalam satu elemen, yaitu prev (kiri) dan next (kanan) sehingga pembacaan data dapat melintas baik maju ke depan, maupun mundur ke belakang. Ilustrasinya yakni sebagai berikut:



Gambar 6. Doubly Linked List

Pada Double Linked List terdapat head dan tail. **Head** adalah sebagai penunjuk data awal, sedangkan **Tail** adalah penunjuk data terakhir.

KELEBIHAN DOUBLE LINKED LIST

Double Linked List memiliki beberapa kelebihan yang menjadikannya unggul dalam situasi tertentu. Contohnya adalah sebagai berikut:

- Operasi menambah node baru di akhir linked list atau menghapus node dari akhir dapat dilakukan tanpa perlu melakukan traversal seperti pada Single Linked List.
- Struktur ini bisa digunakan dalam sistem navigasi yang memerlukan pergerakan maju dan mundur.
- Bisa digunakan untuk mengimplementasikan Tree dalam struktur data.
- Memungkinkan traversal dalam dua arah, baik ke depan maupun ke belakang.
- Memudahkan navigasi, pembacaan, dan penghapusan beberapa node.
- Mendukung akses mundur atau reverse access.
- Penyisipan node dapat dilakukan dengan sangat efisien. Traversal dimulai dari head jika node lebih dekat ke head, dan dari tail jika node lebih dekat ke tail.

KEKURANGAN DOUBLE LINKED LIST

Di sisi lain, Doubly Linked List juga memiliki beberapa kekurangan. Beberapa kekurangan tersebut dapat dijabarkan pada beberapa poin berikut:

- Setiap node dalam Double Linked List memiliki pointer tambahan, sehingga membutuhkan lebih banyak memori.
- Operasi pada Double Linked List memerlukan lebih banyak penggunaan pointer, yang menyebabkan waktu eksekusi menjadi lebih lama.
- Pengaksesan elemen secara acak atau random tidak dapat dilakukan secara langsung.

INISIALISASI DOUBLE LINKED LIST

Adapun algoritma untuk mengimplementasikan Doubly Linked List dapat dilihat pada gambar berikut.

```
1  struct mahasiswa {
2      string nama;
3      int nim;
4      double ipk;
5  };
6
7  struct Node {
8      mahasiswa data;
9      Node *next;
10     Node *prev;
11 };
12
13 Node *head = nullptr, *tail = nullptr;
```

Gambar 7. Struktur Double Linked List

TRAVERSAL

Operasi traversal pada Doubly Linked List dapat dilakukan dengan membuat variabel bantu yang mana akan berisikan nilai dari ***head** atau ***tail**. Kemudian dengan menggunakan variabel bantu tersebut, digunakan perulangan dengan kondisi apabila nilai dari **next** pada variabel bantu bukan **null** dan traversal belum sampai pada index yang diinginkan, maka ubah nilai dari variabel bantu menjadi nilai **next** yang terdapat pada variabel bantu saat ini.

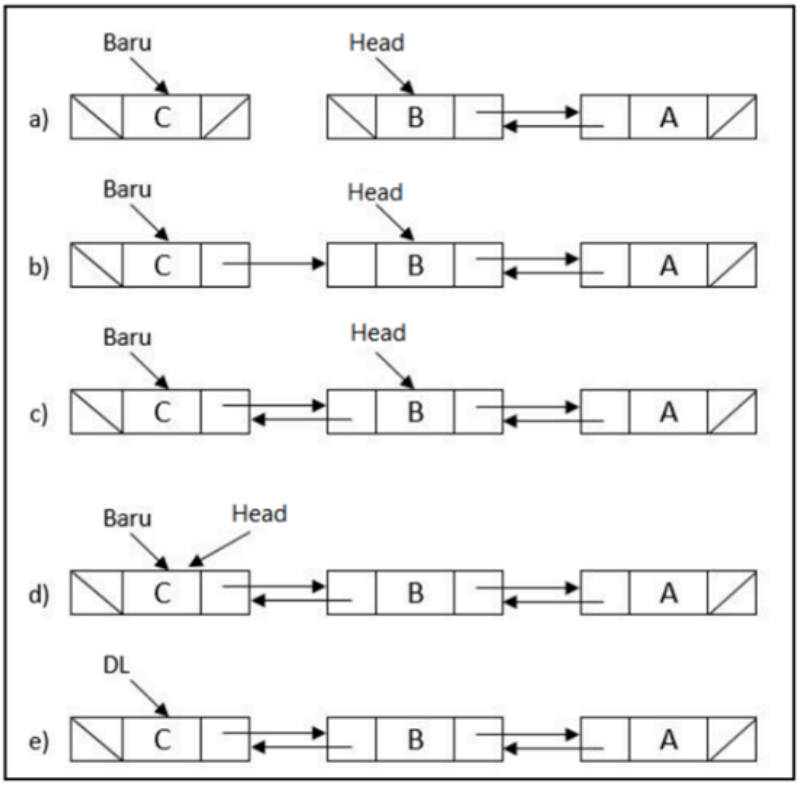
```
#include <iostream>
using namespace std;

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

Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
```

```
    newNode->prev = nullptr;
    return newNode;
}

void insertEnd(Node** head, int value) {
    Node* newNode = createNode(value);
    if (*head == nullptr) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

void traverseForward(Node* head) {
    if (head == nullptr) {
        cout << "List is empty!" << endl;
        return;
    }
    Node* temp = head;
    cout << "Forward Traversal: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void traverseBackward(Node* head) {
    if (head == nullptr) {
        cout << "List is empty!" << endl;
        return;
    }
    // Move to the last node
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    // Traverse backwards
    cout << "Backward Traversal: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int main() {
```

```
    Node* head = nullptr;

    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    insertEnd(&head, 40);

    traverseForward(head);

    traverseBackward(head);

    return 0;
}
```

INSERT FIRST

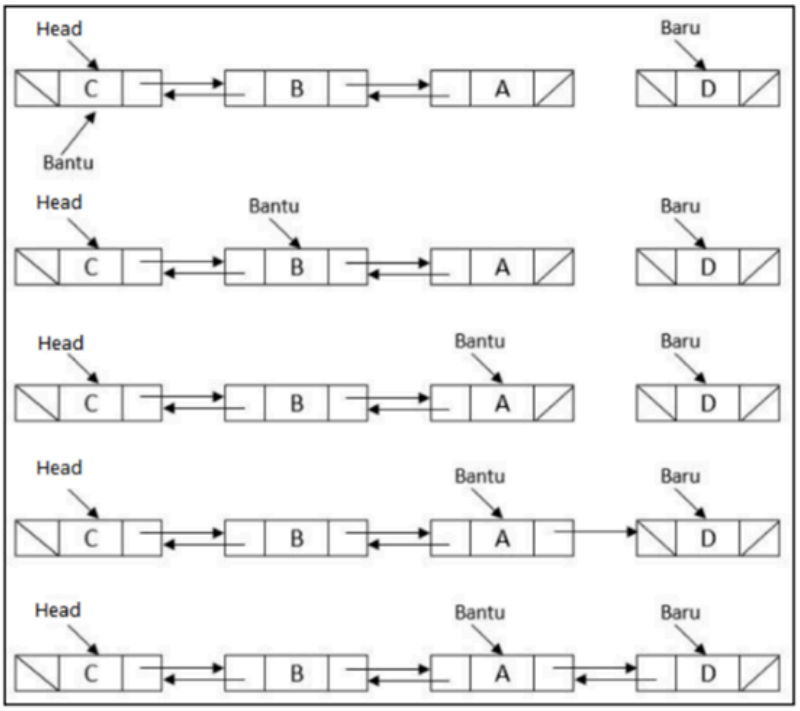
Anggap bahwa telah terdapat sebuah node, apabila kita ingin melakukan operasi ini maka hal yang perlu dilakukan ialah mengubah nilai pada nilai **prev** pada node baru menjadi nilai **null**, setelah itu ubah nilai **next** pada node baru agar mengarah ke node yang saat ini menjadi **head**, langkah berikutnya ialah mengubah nilai **prev** pada node **head** agar mengarah ke node baru. Apabila telah melakukan beberapa langkah tersebut, maka langkah terakhir yang perlu kita lakukan ialah mengubah nilai **head** menjadi node baru.



Gambar 8. Insert First

INSERT LAST

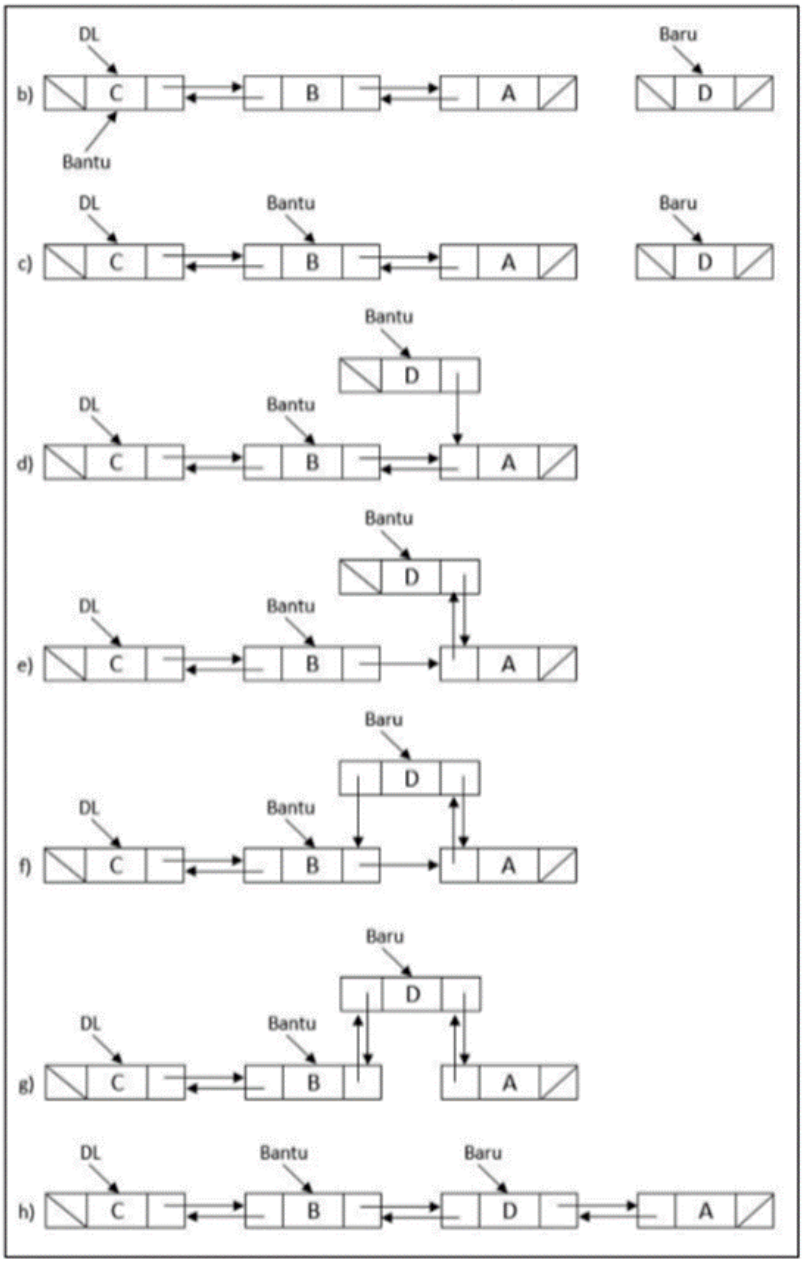
Anggap bahwa sudah terdapat sebuah node. Jika kita ingin melakukan operasi ini, langkah pertama yang perlu dilakukan adalah mengubah nilai **prev** pada node baru agar mengarah ke node yang saat ini menjadi node **tail**. Selanjutnya, ubah nilai **next** pada node baru menjadi **null**. Setelah itu, ubah nilai **next** pada node akhir agar mengarah ke node baru. Terakhir, ubah node **tail** menjadi node baru.



Gambar 9. Insert Last

INSERT SPECIFIC

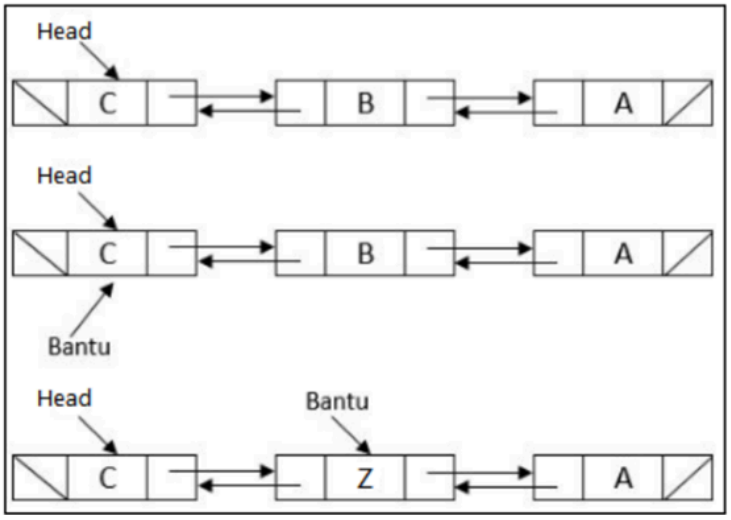
Anggap bahwa sudah terdapat beberapa node, dan kita ingin menyisipkan node baru di posisi tertentu. Langkah pertama yang perlu dilakukan adalah melakukan traversal ke urutan (indeks) node yang ingin disisipkan. Setelah itu, ubah nilai next pada node baru agar mengarah ke node yang saat ini berada setelahnya. Selanjutnya, ubah nilai prev pada node baru agar mengarah ke node yang berada sebelumnya. Setelah langkah tersebut, ubah nilai next pada node sebelumnya agar mengarah ke node baru. Terakhir, ubah nilai prev pada node setelahnya agar mengarah ke node baru.



Gambar 10. Insert Specific

UPDATE DATA

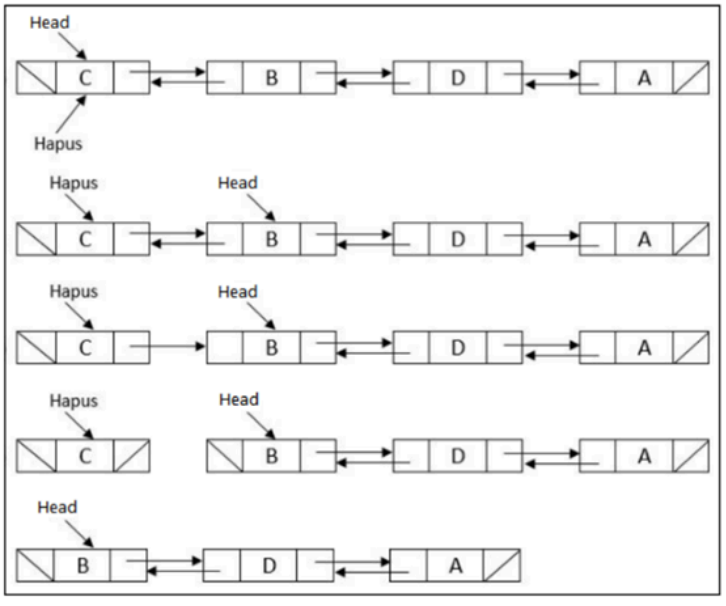
Anggap bahwa sudah terdapat sebuah node, dan kita ingin mengubah data pada node tertentu. Langkah pertama yang perlu dilakukan adalah melakukan traversal hingga mencapai node yang ingin diubah. Setelah itu, ubah nilai data pada node tersebut sesuai dengan nilai yang diinginkan.



Gambar 11. Mengubah Data

DELETE FIRST

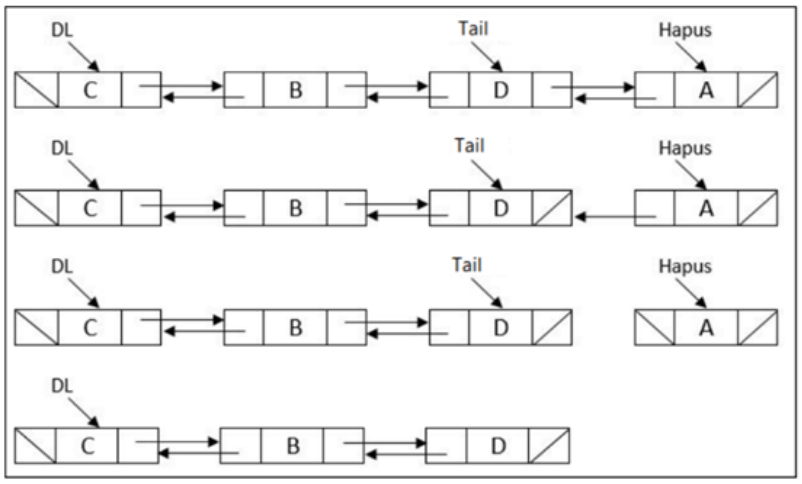
Anggap bahwa sudah terdapat beberapa node, dan kita ingin menghapus node pertama. Langkah pertama yang perlu dilakukan adalah menyimpan alamat node pertama pada variabel sementara. Setelah itu, ubah nilai node **head** agar mengarah ke node setelahnya (alamat dari node setelah node pertama). Selanjutnya, ubah nilai **prev** pada node baru yang saat ini merupakan **head** menjadi **null**. Terakhir, hapus node yang disimpan dalam variabel sementara dengan menggunakan perintah **delete**.



Gambar 12. Delete First

DELETE LAST

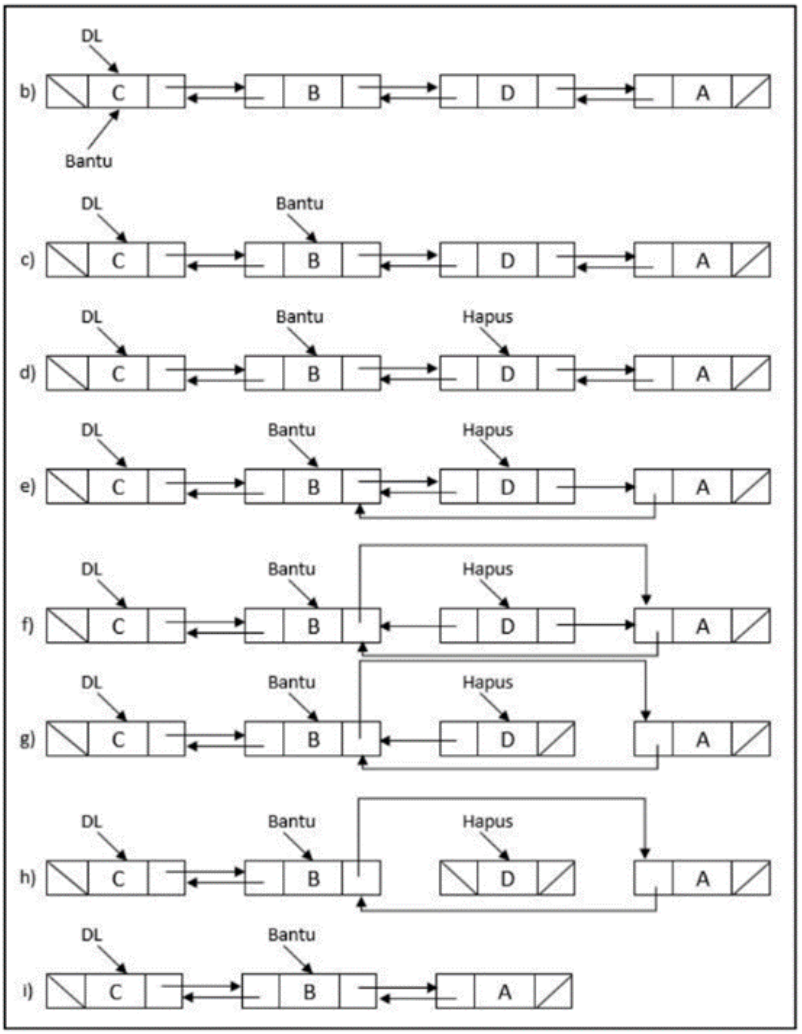
Anggap bahwa sudah terdapat beberapa node, dan kita ingin menghapus node terakhir. Langkah pertama yang perlu dilakukan adalah menyimpan alamat node terakhir pada variabel sementara. Setelah itu, ubah nilai **tail** agar mengarah ke node sebelumnya (alamat dari node sebelum node terakhir). Selanjutnya, ubah nilai **next** pada node baru yang saat ini adalah **tail** menjadi **null**. Terakhir, hapus node yang disimpan dalam variabel sementara dengan menggunakan perintah **delete**.



Gambar 13. Delete Last

DELETE SPECIFIC

Anggap bahwa sudah terdapat beberapa node, dan kita ingin menghapus node tertentu. Langkah pertama yang perlu dilakukan adalah traversal ke node yang ingin dihapus menggunakan variabel bantu. Jika node yang ingin dihapus adalah node awal, maka ubah nilai **head** agar mengarah ke node setelahnya. Jika node yang ingin dihapus adalah node terakhir, ubah nilai **tail** agar mengarah ke node sebelumnya. Apabila terdapat sebuah node sebelum node yang ingin dihapus (misalnya, node A sebelum node B), maka ubah nilai **next** dari node A agar mengarah ke node setelah B. Jika terdapat node setelah node yang ingin dihapus (misalnya, node C setelah node B), ubah nilai **prev** dari node C agar mengarah ke node sebelum B. Setelah itu, kosongkan memori dari node yang ingin dihapus dengan menggunakan perintah **delete**.



Gambar 14. Delete Specific

SOURCE CODE

Adapun source code untuk doubly linked list ini dapat dilihat pada link ini: https://pastebin.com/bPxiuWSF.

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

struct mahasiswa{
    string nama;
    int nim;
    double ipk;
};

struct Node{
```

```
    mahasiswa data;
    Node *next;
    Node *prev;
};

int panjangList = 0;

void show(Node *head){
    if (head == NULL){
        cout << "LinkedList Kosong" << endl;
        system("pause");
        return;
    } else {
        Node *temp = head;
        int indeks = 0;
        while (temp != NULL){
            cout << "[" << indeks << "]" << endl;
            cout << "Nama: " << temp->data.nama << endl;
            cout << "NIM : " << temp->data.nim << endl;
            cout << "IPK : " << temp->data.ipk << endl;
            temp = temp->next;
            indeks++;
        }
        panjangList = indeks;
        system("pause");
    }
}

void insertLast(Node *&head, Node *&tail){
    cout << "\n[ Tambah data di akhir ]" << endl;
    Node *newNode = new Node();
    cout << "Nama: "; cin.ignore(); getline(cin, newNode->data.nama);
    cout << "NIM : "; cin >> newNode->data.nim;
    cout << "IPK : "; cin >> newNode->data.ipk;
    newNode->prev = tail;
    newNode->next = NULL;

    if (head == NULL && tail == NULL){
        head = newNode;
        tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
    panjangList++;
    cout << "Data telah tersimpan." << endl;
    system("pause");
}

void deleteLast(Node *&head, Node *&tail) {
    if (head == NULL && tail == NULL){
        cout << "LinkedList Kosong" << endl;
        system("pause");
```

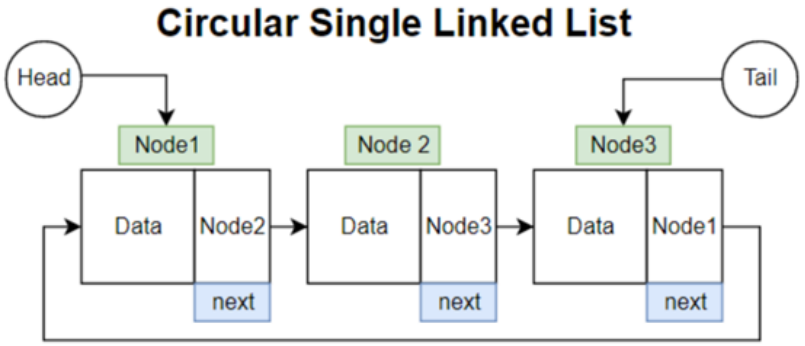
```
    } else if (head->next == NULL){
        delete head;
        head = NULL;
        tail = NULL;
        panjangList--;
        cout << "Data telah terhapus." << endl;
        system("pause");
    } else {
        Node *del = tail;
        tail = tail->prev;
        tail->next = NULL;
        delete del;
        panjangList--;
        cout << "Data telah terhapus." << endl;
        system("pause");
    }
}

int main(){
    Node *HEAD = NULL, *TAIL = NULL;
    int menu = -1;
    while (menu != 0){
        system("cls");
        cout << "\n[ DOUBLE LINKED LIST ]" << endl;
        cout << "Menu:" << endl;
        cout << "1. Tampilkan Data" << endl;
        cout << "2. Tambah Data di Akhir" << endl;
        cout << "3. Hapus Data Terakhir" << endl;
        cout << "0. Exit" << endl;
        cout << "Menu > "; cin >> menu;
        switch(menu) {
            case 1: show(HEAD); break;
            case 2: insertLast(HEAD, TAIL); break;
            case 3: deleteLast(HEAD, TAIL); break;
            case 0: continue;
            default: cout << "ERROR: Menu tidak tersedia." << endl;
system("pause"); break;
        }
    }
    return 0;
}
```

CIRCULAR LINKED LIST

Circular Linked List adalah jenis linked list yang tidak memiliki nilai NULL pada pointer yang menyimpan alamat node sebelumnya dan berikutnya. Circular Linked List bisa diterapkan pada Single Linked List maupun Double Linked List. Dalam struktur ini, tidak ada node yang bernilai NULL karena node terakhir (TAIL) dihubungkan kembali ke node pertama (HEAD) pada Single Linked List, dan pada Double Linked List, node pertama (HEAD) serta node terakhir (TAIL) saling terhubung.

CIRCULAR SINGLE LINKED LIST

Operasi yang dapat dilakukan pada Circular Linked List sama dengan yang ada pada Single Linked List. Perbedaannya terletak pada penghubungan di mana node terakhir (TAIL) harus menunjuk ke node pertama (HEAD).



Gambar 15. Circular Single Linked List

Sedangkan untuk source code mengenai Circular Single Linked List ini dapat dilihat pada link berikut: https://pastebin.com/KQY4MXs6.

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

struct Mahasiswa {
    string nama;
    int nim;
    double ipk;
};

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

Node* newNode() {
    Node *nodeBaru = new Node();
    cout << "Masukan nama : ";
    cin >> nodeBaru->data.nama;
    cout << "Masukan nim  : ";
    cin >> nodeBaru->data.nim;
    cout << "Masukan ipk  : ";
    cin >> nodeBaru->data.ipk;
    nodeBaru->next = nullptr;
    return nodeBaru;
}
```

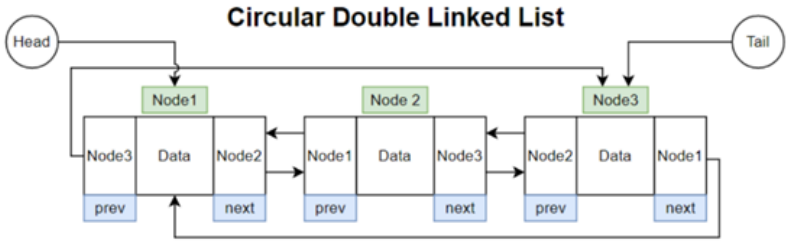
```
void addFirst(Node *&head) {
    Node *nodeBaru = newNode();
    if (head != nullptr) {
        Node *temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        nodeBaru->next = head;
        temp->next = nodeBaru;
        head = nodeBaru;
    } else {
        head = nodeBaru;
        nodeBaru->next = head;
    }
}

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

void display(Node *head) {
    if (head == nullptr) {
        cout << "-----------------------------" << endl;
        cout << "Linked List Kosong" << endl;
        cout << "-----------------------------" << endl;
        return;
    }

    Node *temp = head;
    cout << "-----------------------------" << endl;
    do {
        cout << "Nama: " << temp->data.nama << endl;
        cout << "NIM : " << temp->data.nim << endl;
        cout << "IPK : " << temp->data.ipk << endl;
        cout << "-----------------------------" << endl;
        temp = temp->next;
    } while (temp != head);
}

void deleteNode(Node *&head) {
    if (head == nullptr) {
```

```
        cout << "----------------------------" << endl;
        cout << "Linked List Kosong" << endl;
        cout << "----------------------------" << endl;
        return;
    }

    int key;
    cout << "Masukan nim yang ingin dihapus : ";
    cin >> key;

    // Jika hanya ada 1 node
    if (head->data.nim == key && head->next == head) {
        delete head;
        head = nullptr;
        cout << "Data berhasil dihapus" << endl;
        return;
    }

    Node *temp = head;
    // Jika menghapus node pertama
    if (head->data.nim == key) {
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = head->next;
        delete head;
        head = temp->next;
        return;
    }

    // Menghapus di antara node
    while (temp->next != head && temp->next->data.nim != key) {
        temp = temp->next;
    }

    if (temp->next->data.nim == key) {
        Node *d = temp->next;
        temp->next = d->next;
        delete d;
        cout << "Data berhasil dihapus" << endl;
    } else {
        cout << "NIM tidak ditemukan" << endl;
    }
}

int main() {
    Node *HEAD = nullptr;
    int pilihan;
    while (true) {
        cout << "Program linked list" << endl;
        cout << "1. Add First" << endl;
        cout << "2. Add Last" << endl;
        cout << "3. Delete Node" << endl;
```

```
        cout << "4. Display" << endl;
        cout << "9. Exit Program" << endl;
        cout << "Masukan pilihan : ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                addFirst(HEAD);
                break;
            case 2:
                addLast(HEAD);
                break;
            case 3:
                deleteNode(HEAD);
                break;
            case 4:
                display(HEAD);
                break;
            case 9:
                cout << "Exit Program" << endl;
                return 0;
            default:
                cout << "Pilihan tidak ada" << endl;
        }
    }
    return 0;
}
```

CIRCULAR DOUBLE LINKED LIST

Operasi yang dapat dilakukan pada Circular Linked List mirip dengan yang ada pada Double Linked List. Perbedaannya terletak pada penghubungan, di mana node terakhir (TAIL) harus menunjuk ke node pertama (HEAD), dan node pertama (HEAD) juga harus menunjuk kembali ke node terakhir (TAIL).



Gambar 16. Circular Double Linked List

Sedangkan untuk source code mengenai Circular Single Linked List ini dapat dilihat pada link berikut: https://pastebin.com/E7LTYax8.

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

struct Mahasiswa {
    string nama;
    int nim;
    double ipk;
};

struct Node {
    Mahasiswa data;
    Node *next;
    Node *prev;
};

Node* newNode() {
    Node *nodeBaru = new Node();
    cout << "Masukan nama : ";
    cin >> nodeBaru->data.nama;
    cout << "Masukan nim  : ";
    cin >> nodeBaru->data.nim;
    cout << "Masukan ipk  : ";
    cin >> nodeBaru->data.ipk;
    nodeBaru->next = nodeBaru;
    nodeBaru->prev = nodeBaru;
    return nodeBaru;
}

void addFirst(Node *&head, Node *&tail) {
    Node *nodeBaru = newNode();
    if (head != nullptr) {
        nodeBaru->next = head;
        nodeBaru->prev = tail;
        head->prev = nodeBaru;
        tail->next = nodeBaru;
        head = nodeBaru;
    } else {
        head = nodeBaru;
        tail = nodeBaru;
    }
}

void addLast(Node *&head, Node *&tail) {
    Node *nodeBaru = newNode();
    if (head != nullptr) {
        nodeBaru->prev = tail;
        nodeBaru->next = head;
        tail->next = nodeBaru;
        head->prev = nodeBaru;
        tail = nodeBaru;
    } else {
```

```
        head = nodeBaru;
        tail = nodeBaru;
    }
}

void display(Node *head, bool isHeadToTail) {
    if (head == nullptr) {
        cout << "-----------------------------" << endl;
        cout << "Linked List Kosong" << endl;
        cout << "-----------------------------" << endl;
        return;
    }

    Node *temp = (isHeadToTail) ? head : head->prev;
    cout << "-----------------------------" << endl;

    do {
        cout << "Nama: " << temp->data.nama << endl;
        cout << "NIM : " << temp->data.nim << endl;
        cout << "IPK : " << temp->data.ipk << endl;
        cout << "-----------------------------" << endl;
        temp = (isHeadToTail) ? temp->next : temp->prev;
    } while (temp != head);
}

void deleteNode(Node *&head, Node *&tail) {
    if (head == nullptr) {
        cout << "-----------------------------" << endl;
        cout << "Linked List Kosong" << endl;
        cout << "-----------------------------" << endl;
        return;
    }

    int key;
    cout << "Masukan nim yang ingin dihapus : ";
    cin >> key;

    // Jika hanya ada satu node
    if (head->data.nim == key && head->next == head) {
        delete head;
        head = nullptr;
        tail = nullptr;
        cout << "Data berhasil dihapus" << endl;
        return;
    }

    Node *temp = head;

    // Jika yang ingin dihapus adalah node pertama
    if (head->data.nim == key) {
        tail->next = head->next;
        head->next->prev = tail;
        delete head;
```

```
        head = tail->next;
        return;
    }

    // Menghapus di antara node
    while (temp->next != head && temp->next->data.nim != key) {
        temp = temp->next;
    }

    if (temp->next->data.nim == key) {
        Node *d = temp->next;
        temp->next = d->next;
        d->next->prev = temp;
        delete d;
        cout << "Data berhasil dihapus" << endl;
    } else {
        cout << "NIM tidak ditemukan" << endl;
    }
}

int main() {
    Node *HEAD = nullptr;
    Node *TAIL = nullptr;
    int pilihan;

    while (true) {
        cout << "Program linked list" << endl;
        cout << "1. Add First" << endl;
        cout << "2. Add Last" << endl;
        cout << "3. Delete Node" << endl;
        cout << "4. Display" << endl;
        cout << "9. Exit Program" << endl;
        cout << "Masukan pilihan : ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                addFirst(HEAD, TAIL);
                break;
            case 2:
                addLast(HEAD, TAIL);
                break;
            case 3:
                deleteNode(HEAD, TAIL);
                break;
            case 4: {
                cout << "Display" << endl;
                cout << "1. HEAD to TAIL" << endl;
                cout << "2. TAIL to HEAD" << endl;
                cout << "Masukan pilihan : ";
                cin >> pilihan;
                if (pilihan == 1) {
                    display(HEAD, true);
```

```
                } else if (pilihan == 2) {
                    display(HEAD, false);
                } else {
                    cout << "Pilihan tidak ada" << endl;
                }
                break;
            }
            case 9:
                cout << "Exit Program" << endl;
                return 0;
            default:
                cout << "Pilihan tidak ada" << endl;
        }
    }
    return 0;
}
```