

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 3
SINGLE AND DOUBLE LINKED LIST**



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

**Disusun Oleh :
HAIKAL SATRIATAMA (2311102066)
IF-11-B**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

A. Dasar Teori

1. Single Linked List

Single Linked List adalah sebuah struktur data linear yang tersusun atas simpul-simpul (atau node) yang terhubung secara sekuensial. Setiap simpul hanya memiliki satu pointer yang menunjuk ke simpul berikutnya dalam urutan. Ini adalah salah satu bentuk paling sederhana dari linked list. Kelebihan: Menggunakan memori dengan efisien karena hanya memerlukan satu pointer per simpul.

Operasi dasar

- **Penyisipan (Insertion):** Data dapat dimasukkan di awal, tengah, atau akhir linked list dengan mengubah pointer **next**.
- **Penghapusan (Deletion):** Simpul dapat dihapus dengan mengatur ulang pointer **next**.
- **Penelusuran (Traversal):** Linked list dapat diiterasi dari awal hingga akhir dengan menggunakan pointer **next**.
- **Pencarian (Searching):** Pencarian data dapat dilakukan dengan menelusuri linked list hingga menemukan data yang sesuai.

2. Double Linked List

Double linked list adalah suatu bentuk linked list di mana setiap simpul memiliki dua pointer: **next** yang menunjuk ke simpul berikutnya, dan **prev** yang menunjuk ke simpul sebelumnya. Dengan konfigurasi ini, traversal dari depan ke belakang dan sebaliknya memungkinkan dalam double linked list. Kelebihan: Dapat melakukan traversal maju dan mundur dengan mudah karena setiap simpul memiliki pointer prev dan next.

Operasi dasar:

- Penyisipan (Insertion): Data dapat dimasukkan di awal, tengah, atau akhir linked list dengan mengubah pointer next dan prev.
- Penghapusan (Deletion): Simpul dapat dihapus dengan mengatur ulang pointer next dan prev.
- Penelusuran (Traversal): Linked list dapat diiterasi dari awal hingga akhir, atau sebaliknya, menggunakan pointer next dan prev.
- Pencarian (Searching): Pencarian data dapat dilakukan dengan menelusuri linked list hingga menemukan data yang sesuai. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

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

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

Node *head;
Node *tail;

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

// Pengecekan
bool isEmpty() {
    return head == NULL;
}

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

```

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

// Hitung Jumlah List
int hitungList() {
    Node *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;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        if (head->next != NULL) {

```

```

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

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

```

```

}

// Hapus Tengah
void hapusTengah(int posisi) {
    if (posisi < 1 || posisi > hitungList()) {
        cout << "Posisi di luar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    } else {
        Node *hapus, *bantu, *bantu2;
        int nomor = 1;
        bantu = head;
        while (nomor <= posisi) {
            if (nomor == posisi - 1) {
                bantu2 = bantu;
            }
            if (nomor == posisi) {
                hapus = bantu;
            }
            bantu = bantu->next;
            nomor++;
        }
        bantu2->next = bantu;
        delete hapus;
    }
}

// Ubah Depan
void ubahDepan(int data) {
    if (!isEmpty()) {

```



```

        head->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah Tengah
void ubahTengah(int data, int posisi) {
    if (!isEmpty()) {
        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;
            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()) {

```

```

        tail->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

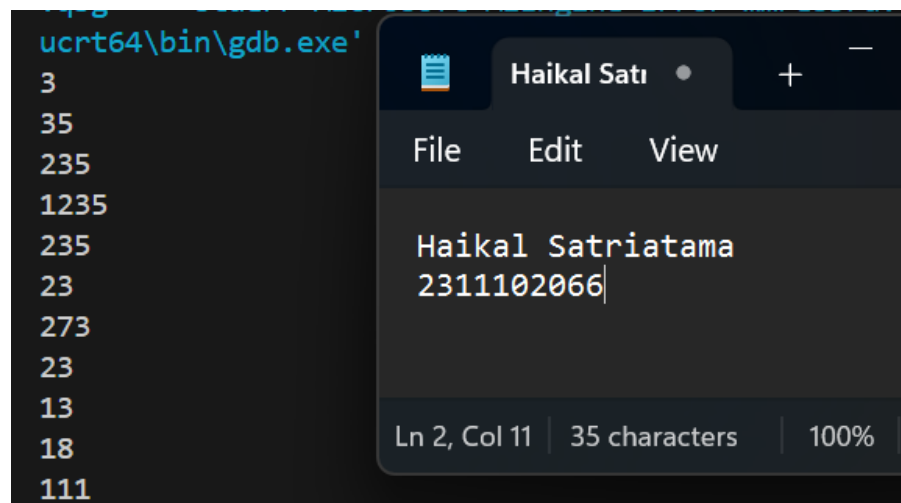
// Hapus List
void clearList() {
    Node *bantu, *hapus;
    bantu = head;
    while (bantu != NULL) {
        hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = tail = NULL;
    cout << "List berhasil terhapus!" << endl;
}

// Tampilkan List
void tampil() {
    Node *bantu = head;
    if (!isEmpty()) {
        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;  
}
```

Screenshots Output



The screenshot shows a debugger window with a list of memory addresses on the left: 3, 35, 235, 1235, 235, 23, 273, 23, 13, 18, and 111. To the right, a text editor window titled 'Haikal Satri' is open, displaying the text 'Haikal Satriatama' followed by '2311102066' on the next line. The status bar at the bottom of the text editor indicates 'Ln 2, Col 11 | 35 characters | 100%'.

Deskripsi

- **Deklarasi Struct Node dan Pointer Head serta Tail**
 - Struktur **Node** digunakan untuk merepresentasikan simpul dalam linked list.
 - Pointer **head** dan **tail** digunakan untuk menandai awal dan akhir dari linked list.
- **Inisialisasi Linked List**
 - Fungsi **init()** digunakan untuk menginisialisasi linked list dengan mengatur **head** dan **tail** menjadi **NULL**.
- **Pengecekan Linked List Kosong**
 - Fungsi **isEmpty()** digunakan untuk memeriksa apakah linked list kosong atau tidak.
- **Penambahan Elemen di Depan Linked List**
 - Fungsi **insertDepan(int nilai)** digunakan untuk menambahkan elemen baru di depan linked list.
- **Penambahan Elemen di Belakang Linked List**
 - Fungsi **insertBelakang(int nilai)** digunakan untuk menambahkan elemen baru di belakang linked list.
- **Perhitungan Jumlah Elemen dalam Linked List**

- Fungsi **hitungList()** digunakan untuk menghitung jumlah elemen dalam linked list.
- **Penambahan Elemen di Tengah Linked List**
 - Fungsi **insertTengah(int data, int posisi)** digunakan untuk menambahkan elemen baru di tengah linked list.
- **Penghapusan Elemen di Depan Linked List**
 - Fungsi **hapusDepan()** digunakan untuk menghapus elemen pertama dari linked list.
- **Penghapusan Elemen di Belakang Linked List**
 - Fungsi **hapusBelakang()** digunakan untuk menghapus elemen terakhir dari linked list.
- **Penghapusan Elemen di Tengah Linked List**
 - Fungsi **hapusTengah(int posisi)** digunakan untuk menghapus elemen di tengah linked list berdasarkan posisi.
- **Pengubahan Elemen di Depan Linked List**
 - Fungsi **ubahDepan(int data)** digunakan untuk mengubah nilai elemen pertama dari linked list.
- **Pengubahan Elemen di Belakang Linked List**
 - Fungsi **ubahBelakang(int data)** digunakan untuk mengubah nilai elemen terakhir dari linked list.
- **Pengubahan Elemen di Tengah Linked List**
 - Fungsi **ubahTengah(int data, int posisi)** digunakan untuk mengubah nilai elemen di tengah linked list berdasarkan posisi.
- **Penghapusan Seluruh Elemen Linked List**
 - Fungsi **clearList()** digunakan untuk menghapus semua elemen dari linked list.
- **Menampilkan Elemen Linked List**
 - Fungsi **tampil()** digunakan untuk menampilkan semua elemen dari linked list.
- **Fungsi Utama (Main)**

- Di dalam fungsi utama (**main()**), beberapa operasi dasar pada linked list seperti penyisipan, penghapusan, pengubahan, dan penampilan dilakukan untuk menguji fungsionalitas dari implementasi linked list.

Guided 2

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    void push(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr) {
            head->prev = newNode;
        } else {
```

```

        tail = newNode;
    }
    head = newNode;
}

void pop() {
    if (head == nullptr) {
        return;
    }

    Node* temp = head;
    head = head->next;

    if (head != nullptr) {
        head->prev = nullptr;
    } else {
        tail = nullptr;
    }
    delete temp;
}

bool update(int oldData, int newData) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == oldData) {
            current->data = newData;
            return true;
        }
        current = current->next;
    }
}

```



```

        return false;
    }

    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

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

int main() {

    DoublyLinkedList list;

    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
    }
}

```

```
cout << "3. Update data" << endl;
cout << "4. Clear data" << endl;
cout << "5. Display data" << endl;
cout << "6. Exit" << endl;

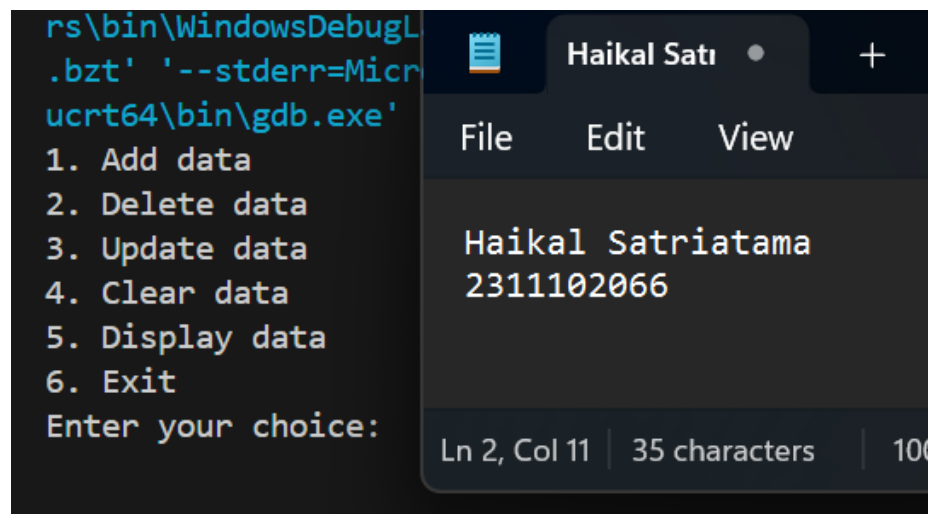
int choice;
cout << "Enter your choice: ";
cin >> choice;

switch (choice) {
    case 1: {
        int data;
        cout << "Enter data to add: ";
        cin >> data;
        list.push(data);
        break;
    }
    case 2: {
        list.pop();
        break;
    }
    case 3: {
        int oldData, newData;
        cout << "Enter old data: ";
        cin >> oldData;
        cout << "Enter new data: ";
        cin >> newData;
        bool updated = list.update(oldData, newData);
        if (!updated) {
            cout << "Data not found" << endl;
```

```
        }
        break;
    }
    case 4: {
        list.deleteAll();
        break;
    }
    case 5: {
        list.display();
        break;
    }
    case 6: {
        return 0;
    }
    default: {
        cout << "Invalid choice" << endl;
        break;
    }
}

return 0;
}
```

Screenshots Output



The screenshot shows a terminal window with a menu and a text editor. The terminal window has a title bar with the text 'rs\bin\WindowsDebugL' and a menu bar with 'File', 'Edit', and 'View'. The terminal content shows a menu with six options: 1. Add data, 2. Delete data, 3. Update data, 4. Clear data, 5. Display data, and 6. Exit. Below the menu is the prompt 'Enter your choice:'. The text editor window has a title bar with the text 'Haikal Satri' and a menu bar with 'File', 'Edit', and 'View'. The text editor content shows the text 'Haikal Satriatama' and '2311102066'. The status bar at the bottom of the text editor shows 'Ln 2, Col 11 | 35 characters | 100%'.

```
rs\bin\WindowsDebugL
.bzt' '--stderr=Micro
ucrt64\bin\gdb.exe'
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

```
Haikal Satri
Haikal Satriatama
2311102066
Ln 2, Col 11 | 35 characters | 100%
```

Deskripsi:

Kelas **Node** didefinisikan untuk merepresentasikan simpul dalam Doubly Linked List. Setiap simpul memiliki tiga anggota data: **data** yang menyimpan nilai data, **prev** yang merupakan pointer ke simpul sebelumnya, dan **next** yang merupakan pointer ke simpul berikutnya.

Kelas **DoublyLinkedList** mendefinisikan operasi-operasi dasar pada Doubly Linked List. Ini termasuk operasi untuk menambah data (**push()**), menghapus data (**pop()**), mengupdate data (**update()**), menghapus semua data (**deleteAll()**), dan menampilkan data (**display()**).

Di dalam fungsi **main()**, program menampilkan menu operasi-operasi yang dapat dilakukan pada Doubly Linked List, yaitu:

1. Menambahkan data baru.
2. Menghapus data.
3. Memperbarui data.
4. Menghapus semua data.
5. Menampilkan semua data.

6. Keluar dari program.

Setelah pengguna memilih operasi yang diinginkan, program menggunakan **switch** statement untuk mengeksekusi operasi yang sesuai. Misalnya, jika pengguna memilih untuk menambahkan data baru, program meminta input data baru dari pengguna dan menambahkannya ke Doubly Linked List menggunakan metode **push()**. Begitu seterusnya untuk setiap operasi yang dipilih oleh pengguna.

- B. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

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

struct Node {
    string nama;
    int umur;
    Node* next;
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    // Function to insert data at the beginning of the
    list
    void insertDepan(string nama, int umur) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->umur = umur;
        newNode->next = head;
        head = newNode;
    }

    // Function to insert data at the end of the list
    void insertBelakang(string nama, int umur) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->umur = umur;
        newNode->next = nullptr;

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

```

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

    // Function to insert data after a specific node
    void insertTengah(string nama, int umur, string
keyNama) {
        Node* temp = head;
        while (temp != nullptr && temp->nama !=
keyNama) {
            temp = temp->next;
        }
        if (temp == nullptr) {
            cout << "Nama " << keyNama << " tidak
ditemukan.\n";
            return;
        }

        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->umur = umur;
        newNode->next = temp->next;
        temp->next = newNode;
    }

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

        if (temp != nullptr && temp->nama == nama) {
            head = temp->next;
            delete temp;
            return;
        }

        while (temp != nullptr && temp->nama != nama)
        {
            prev = temp;
            temp = temp->next;
        }
    }

```

```

        if (temp == nullptr) {
            cout << "Data dengan nama " << nama << "
tidak ditemukan.\n";
            return;
        }

        prev->next = temp->next;
        delete temp;
    }

    // Function to display all data in the list
    void display() {
        Node* temp = head;
        cout << "Nama\tumur\n";
        while (temp != nullptr) {
            cout << temp->nama << "\t" << temp->umur
<< endl;
            temp = temp->next;
        }
    }
};

int main() {

    LinkedList list;
    list.insertDepan("Haikal", 19);
    list.insertBelakang("John", 19);
    list.insertBelakang("Jane", 20);
    list.insertBelakang("Michael", 18);
    list.insertBelakang("Yusuke", 19);
    list.insertBelakang("Akechi", 20);
    list.insertBelakang("Hoshino", 18);
    list.insertBelakang("Karin", 18);

    cout << "Data awal:\n";
    list.display();

    cout << "\nHapus data Akechi:\n";
    list.deleteNode("Akechi");
    list.display();

    cout << "\nTambahkan data Futaba setelah John:\n";
    list.insertTengah("Futaba", 18, "John");
    list.display();
}

```



```

        cout << "\nTambahkan data Igor di awal:\n";
        list.insertDepan("Igor", 20);
        list.display();

        cout << "\nUbah data Michael menjadi Reyn:\n";
        list.deleteNode("Michael");
        list.insertBelakang("Reyn", 18);
        list.display();

        return 0;
    }

```

Screenshots Output

```

ucrt64\bin\gdb
Data awal:
Nama    umur
Haikal  19
John    19
Jane    20
Michael 18
Yusuke  19
Akechi  20
Hoshino 18
Karin   18

```

Haikal Satriatama
2311102066

Ln 2, Col 11 | 35 characters | 100% | Window

```

Hapus data Akechi:
Nama    umur
Haikal  19
John    19
Jane    20
Michael 18
Yusuke  19
Hoshino 18
Karin   18

```

Tambahkan data Futaba setelah John:

```

Nama    umur
Haikal  19
John    19
Futaba  18
Jane    20
Michael 18
Yusuke  19
Hoshino 18
Karin   18

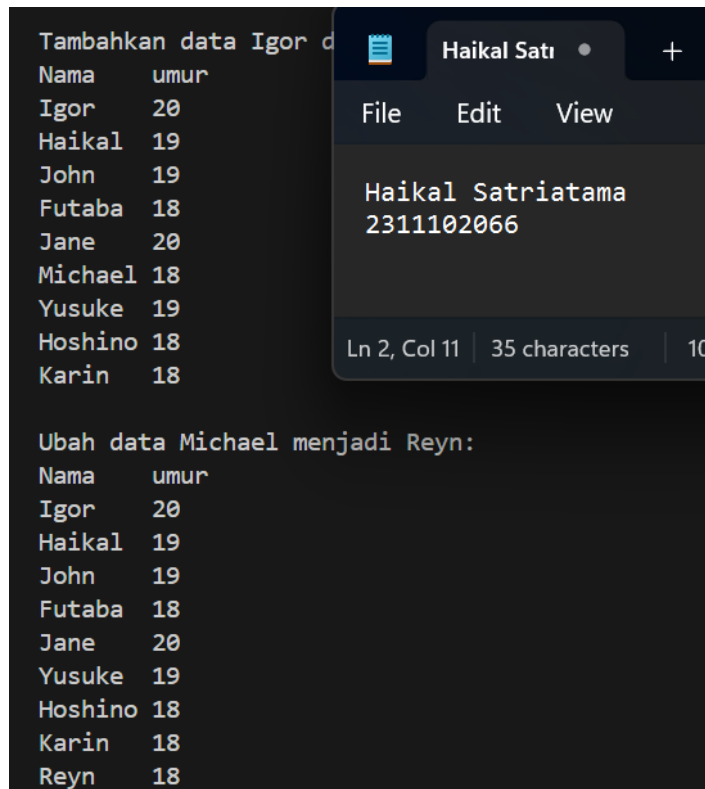
```

Haikal Satriatama
2311102066

Ln 2, Col 11 | 35 characters | 100% | Window

```
Tambahkan data Igor d
Nama    umur
Igor    20
Haikal  19
John    19
Futaba  18
Jane    20
Michael 18
Yusuke  19
Hoshino 18
Karin   18

Ubah data Michael menjadi Reyn:
Nama    umur
Igor    20
Haikal  19
John    19
Futaba  18
Jane    20
Yusuke  19
Hoshino 18
Karin   18
Reyn    18
```



Kelas **LinkedList** memiliki fungsi-fungsi untuk melakukan operasi-operasi dasar pada linked list, termasuk penambahan data di depan (**insertDepan()**), di belakang (**insertBelakang()**), dan setelah simpul tertentu (**insertTengah()**), serta penghapusan data berdasarkan nama (**deleteNode()**) dan penampilan semua data (**display()**).

Setiap simpul dalam linked list direpresentasikan oleh struktur **Node**, yang menyimpan informasi nama dan umur serta pointer yang menunjukkan ke simpul berikutnya. Ketika data dimasukkan, fungsi-fungsi **insertDepan()**, **insertBelakang()**, dan **insertTengah()** akan membuat simpul baru dengan informasi yang sesuai, lalu menambahkannya ke linked list sesuai dengan lokasi yang diinginkan. Sedangkan fungsi **deleteNode()** akan mencari simpul dengan nama yang sesuai, kemudian menghapusnya dari linked list.

Dalam fungsi **main()**, dilakukan serangkaian operasi untuk menguji fungsionalitas dari linked list yang telah dibuat. Data awal dimasukkan ke dalam linked list, kemudian dilakukan penghapusan, penambahan di tengah, penambahan di depan, serta penggantian data, dan hasilnya ditampilkan.

Unguided 2

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

class Barang {
public:
    string namaProduk;
    int harga;
    Barang* prev;
    Barang* next;
};

class DoubleLinkedList {
public:
    Barang* head;
    Barang* tail;

    DoubleLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    void tambah(string namaProduk, int harga) {
        Barang* newNode = new Barang;
        newNode->namaProduk = namaProduk;
        newNode->harga = harga;
        newNode->prev = nullptr;
        newNode->next = head;
```

```

        if (head != nullptr) {
            head->prev = newNode;
        } else {
            tail = newNode;
        }
        head = newNode;
    }

```

```

    void tambahSetelah(string namaProduk, int harga, string
keyNamaProduk) {
        Barang* current = head;
        while (current != nullptr && current->namaProduk !=
keyNamaProduk) {
            current = current->next;
        }

```

```

        if (current == nullptr) {
            cout << "Produk " << keyNamaProduk << " tidak
ditemukan.\n";
            return;
        }

```

```

        Barang* newNode = new Barang;
        newNode->namaProduk = namaProduk;
        newNode->harga = harga;
        newNode->prev = current;
        newNode->next = current->next;

```

```

        if (current->next != nullptr) {
            current->next->prev = newNode;
        } else {

```

```

        tail = newNode;
    }
    current->next = newNode;
}

void hapus(string namaProduk) {
    Barang* current = head;
    while (current != nullptr && current->namaProduk !=
namaProduk) {
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Produk " << namaProduk << " tidak
ditemukan.\n";
        return;
    }

    if (current->prev != nullptr) {
        current->prev->next = current->next;
    } else {
        head = current->next;
    }

    if (current->next != nullptr) {
        current->next->prev = current->prev;
    } else {
        tail = current->prev;
    }
}

```

```

        delete current;
    }

    bool perbarui(string oldNamaProduk, string
newNamaProduk, int newHarga) {
        Barang* current = head;
        while (current != nullptr && current->namaProduk !=
oldNamaProduk) {
            current = current->next;
        }

        if (current == nullptr) {
            cout << "Produk " << oldNamaProduk << " tidak
ditemukan.\n";
            return false;
        }

        current->namaProduk = newNamaProduk;
        current->harga = newHarga;
        return true;
    }

void hapusSemua() {
    Barang* current = head;
    while (current != nullptr) {
        Barang* temp = current;
        current = current->next;
        delete temp;
    }
    head = nullptr;
    tail = nullptr;
}

```

```

    }

    void tampilkan() {
        Barang* current = head;
        cout << left << setw(20) << "Nama Produk" << setw(10)
        << "Harga" << endl;
        while (current != nullptr) {
            cout << left << setw(20) << current->namaProduk
            << setw(10) << current->harga << endl;
            current = current->next;
        }
        cout << endl;
    }
};

int main() {

    DoubleLinkedList list;
    list.tambah("Originote", 60000);
    list.tambah("Somethinc", 150000);
    list.tambah("Skintific", 100000);
    list.tambah("Wardah", 50000);
    list.tambah("Hanasui", 30000);

    while (true) {
        cout << "Toko Skincare Purwokerto" << endl;
        cout << "1. Tambah Data" << endl;
        cout << "2. Hapus Data" << endl;
        cout << "3. Perbarui Data" << endl;
        cout << "4. Tambah Data Urutan Tertentu" << endl;
    }
}

```



```
    cout << "5. Hapus Data Urutan Tertentu" << endl;
    cout << "6. Hapus Seluruh Data" << endl;
    cout << "7. Tampilkan Data" << endl;
    cout << "8. Exit" << endl;
    cout << endl;

    int choice;
    cout << "Masukkan your choice: ";
    cin >> choice;
    cout << endl;

    switch (choice) {
        case 1: {
            string namaProduk;
            int harga;
            cout << "Masukkan nama produk: ";
            cin >> namaProduk;
            cout << "Masukkan harga: ";
            cin >> harga;
            list.tambah(namaProduk, harga);
            break;
        }
        case 2: {
            string namaProduk;
            cout << "Masukkan nama produk yang akan
dihapus: ";

            cin >> namaProduk;
            list.hapus(namaProduk);
            break;
        }
    }
```

```

        case 3: {
            string oldNamaProduk, newNamaProduk;
            int newHarga;

            cout << "Masukkan nama produk yang akan
diperbarui: ";

            cin >> oldNamaProduk;
            cout << "Masukkan nama produk baru: ";
            cin >> newNamaProduk;
            cout << "Masukkan harga baru: ";
            cin >> newHarga;

            bool updated = list.perbarui(oldNamaProduk,
newNamaProduk, newHarga);
            if (!updated) {
                cout << "Data tidak ditemukan" << endl;
            }
            break;
        }
        case 4: {
            string namaProduk, keyNamaProduk;
            int harga;

            cout << "Masukkan nama produk yang akan
ditambahkan setelahnya: ";

            cin >> keyNamaProduk;
            cout << "Masukkan nama produk baru: ";
            cin >> namaProduk;
            cout << "Masukkan harga: ";
            cin >> harga;

            list.tambahSetelah(namaProduk, harga,
keyNamaProduk);

            break;
        }
    }

```

```

        case 5: {
            string namaProduk;
            cout << "Masukkan nama produk yang akan
dihapus: ";

            cin >> namaProduk;
            list.hapus(namaProduk);
            break;
        }
        case 6: {
            list.hapusSemua();
            cout << "Semua data telah dihapus." << endl;
            break;
        }
        case 7: {
            cout << "Data Produk:\n";
            list.tampilkan();
            break;
        }
        case 8: {
            return 0;
        }
        default: {
            cout << "Pilihan tidak valid" << endl;
            break;
        }
    }

    return 0;
}

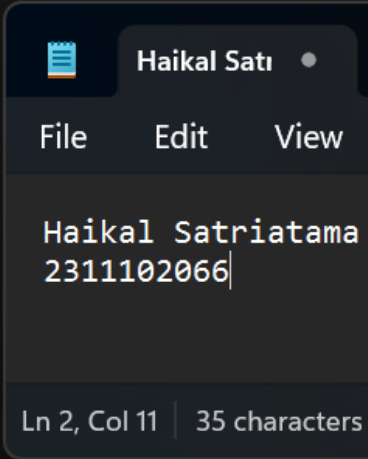
```

Screenshots Output

```
ucrt64\bin\gdb.exe' '--interpreter=mi'
Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Perbarui Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit

Masukkan your choice: 7

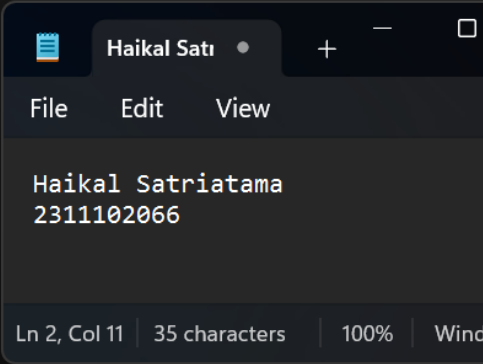
Data Produk:
Nama Produk      Harga
Hanasui          30000
Wardah           50000
Skintific        100000
Somethinc        150000
Originote        60000
```



```
Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Perbarui Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit

Masukkan your choice: 4

Masukkan nama produk yang akan ditambahkan setelahnya: Skintific
Masukkan nama produk baru: Azarine
Masukkan harga: 65000
```



```
Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Perbarui Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit

Masukkan your choice: 2

Masukkan nama produk yang akan dihapus: Wardah
```

Haikal Satri

File Edit View

Haikal Satriatama
2311102066

Ln 2, Col 11 | 35 characters

```
Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Perbarui Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit

Masukkan your choice: 3

Masukkan nama produk yang akan diperbarui: Hanasui
Masukkan nama produk baru: Cleora
Masukkan harga baru: 55000
```

Haikal Satri

File Edit View

Haikal Satriatama
2311102066

Ln 2, Col 11 | 35 characters

```
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit

Masukkan your choice: 7

Data Produk:
Nama Produk      Harga
Cleora            55000
Skintific         100000
Azarine           65000
Somethinc         150000
Originote         60000
```

Haikal Satri

File Edit View

Haikal Satriatama
2311102066

Ln 2, Col 11 | 35 characters

Deskripsi:

Setiap barang direpresentasikan sebagai objek dari kelas **Barang**, yang memiliki atribut nama produk dan harga. Kelas **DoubleLinkedList** memiliki fungsi-fungsi untuk menambah, menghapus, memperbarui, dan menampilkan data barang.

Dalam implementasi ini, setiap elemen dalam double linked list direpresentasikan oleh objek **Barang**. Setiap objek **Barang** memiliki dua pointer, yaitu **prev** dan **next**, yang menunjukkan elemen sebelumnya dan selanjutnya dalam double linked list. Fungsi **tambah** digunakan untuk menambahkan barang baru ke awal double linked list, **tambahSetelah** digunakan untuk menambahkan barang baru setelah sebuah barang tertentu, **hapus** digunakan untuk menghapus barang berdasarkan nama produknya, **perbarui** digunakan untuk memperbarui informasi barang berdasarkan nama produknya, dan **tampilkan** digunakan untuk menampilkan semua data barang yang ada dalam double linked list.

Pada fungsi **main**, terdapat sebuah loop yang memungkinkan pengguna untuk memilih berbagai operasi yang ingin dilakukan terhadap data barang, seperti menambah, menghapus, memperbarui, atau menampilkan data barang. Pengguna diminta untuk memasukkan pilihan operasi yang diinginkan, dan sesuai dengan pilihan tersebut, fungsi yang sesuai akan dipanggil dari objek **DoubleLinkedList**.

C. Kesimpulan

Single linked list adalah struktur data di mana setiap node memiliki satu pointer yang menunjukkan ke node berikutnya dalam list. Ini membuat penambahan dan penghapusan elemen di ujung depan list menjadi efisien, tetapi pencarian atau akses elemen di bagian tengah list membutuhkan traversal dari awal list.

Sementara itu, double linked list adalah struktur data di mana setiap node memiliki dua pointer: satu yang menunjukkan ke node berikutnya dan satu lagi yang menunjukkan ke node sebelumnya dalam list. Hal ini memungkinkan traversal maju dan mundur dalam list dengan efisien, namun membutuhkan lebih banyak ruang memori karena adanya pointer tambahan.

Keuntungan utama dari penggunaan linked list adalah kemampuan untuk menambah dan menghapus elemen dengan cepat tanpa perlu melakukan realokasi memori seperti yang terjadi pada array. Namun, penggunaan linked list mungkin kurang efisien dalam hal pengaksesan acak elemen dan membutuhkan lebih banyak ruang memori untuk menyimpan pointer tambahan.

D. Referensi

Rassokhin, D. (2020). The C++ programming language in cheminformatics and computational chemistry. *Journal of Cheminformatics*, 12(10). Retrieved from *Journal of Cheminformatics*.

Meidyan Permata Putri, Guntoro Barovih, Rezanía Agramanisti Azdy, Yuniansyah, Andri Saputra, Yesi Sriyeni, Arsia Rini, Fadhila Tangguh Admojo. *Algoritma dan Struktur Data*. Widina Bhakti Persada, 2022. ISBN: 978-623-459-182-8.