

LAPORAN PRAKTIKUM
Modul IV
“Single Linked List”



Disusun Oleh:
Zivana Afra Yulianto -2211104039
SE-07-02

Dosen:
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

Tujuan dari praktikum ini adalah untuk memahami dan mengimplementasikan struktur data Single Linked List menggunakan bahasa pemrograman C++. Praktikum ini bertujuan untuk:

- Mempelajari cara membuat, mengelola, dan memanipulasi linked list.
- Mengimplementasikan operasi dasar seperti penambahan, penghapusan, pencarian, dan penghitungan elemen dalam linked list.
- Memahami pengelolaan memori secara manual dalam konteks struktur data dinamis.

2. Landasan Teori

Linked List adalah struktur data dinamis yang terdiri dari sekumpulan elemen (node), di mana setiap elemen berisi data dan pointer (atau referensi) ke elemen berikutnya. Linked List memiliki beberapa keunggulan dibandingkan array, antara lain:

- **Dinamisme:** Linked List dapat bertambah atau berkurang ukurannya sesuai kebutuhan, tanpa batasan ukuran tetap seperti array.
- **Efisiensi dalam Operasi:** Operasi penyisipan dan penghapusan dapat dilakukan dengan lebih efisien dibandingkan array, terutama jika dilakukan di awal atau akhir list.
- **Pengelolaan Memori:** Linked List memungkinkan penggunaan memori yang lebih fleksibel, karena alokasi dan dealokasi memori dapat dilakukan saat runtime.

Ada beberapa jenis Linked List, termasuk:

- **Single Linked List:** Setiap node memiliki satu pointer yang mengarah ke node berikutnya.
- **Doubly Linked List:** Setiap node memiliki dua pointer, satu untuk node berikutnya dan satu untuk node sebelumnya.
- **Circular Linked List:** Node terakhir mengarah kembali ke node pertama, membentuk siklus.

3. Guided

A. GUIDED 1

Code :

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

// Deklarasi Struct untuk mahasiswa
struct mahasiswa
{
    char nama[30];
    char nim[10];
};

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

Node *head;
Node *tail;

// Inisialisasi List
void init()
{
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty()
{
    return head == nullptr;
}

// Tambah Depan
void insertDepan(const mahasiswa &data)
{
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty())
    {
        head = tail = baru;
    }
    else
    {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(const mahasiswa &data)
{
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty())
    {
        head = tail = baru;
    }
    else
    {
        tail->next = baru;
        tail = baru;
    }
}

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

```

// Hapus Depan
void hapusDepan()
{
    if (!isEmpty())
    {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr)
        {
            tail = nullptr; // Jika list menjadi kosong
        }
    }
    else
    {
        cout << "List kosong!" << endl;
    }
}

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

// Tampilkan List
void tampil()
{
    Node *current = head;
    if (!isEmpty())
    {
        while (current != nullptr)
        {
            cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
            current = current->next;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
    cout << endl;
}

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

// Main function
int main()
{
    init();
}

```

```

// Contoh data mahasiswa
mahasiswa m1 = {"Alice", "123456"};
mahasiswa m2 = {"Bob", "654321"};
mahasiswa m3 = {"Charlie", "112233"};

// Menambahkan mahasiswa ke dalam list
insertDepan(m1);
tampil();
insertBelakang(m2);
tampil();
insertDepan(m3);
tampil();

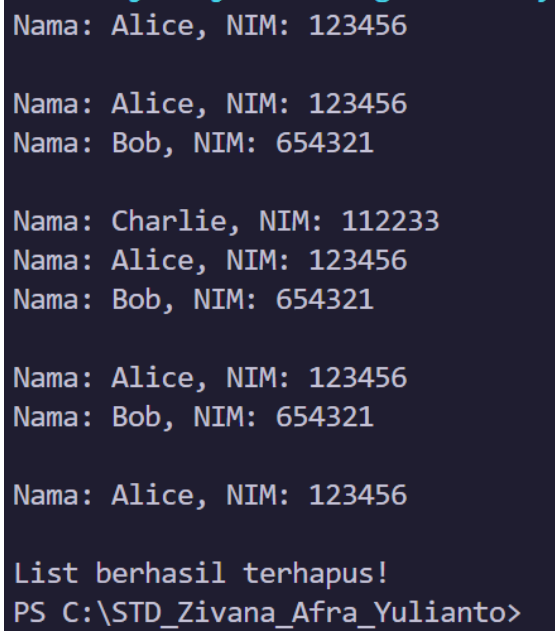
// Menghapus elemen dari list
hapusDepan();
tampil();
hapusBelakang();
tampil();

// Menghapus seluruh list
clearList();

return 0;
}

```

Screenshoot Output :



```

Nama: Alice, NIM: 123456

Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

Nama: Alice, NIM: 123456

List berhasil terhapus!
PS C:\STD_Zivana_Afra_Yulianto>

```

Deskripsi :

Program C++ ini mengimplementasikan **Single Linked List** untuk menyimpan data mahasiswa (nama dan NIM). Program mendukung operasi:

1. **Tambah Depan/Belakang:** Menambahkan node baru di awal/akhir list.
2. **Hapus Depan/Belakang:** Menghapus node dari awal/akhir list.
3. **Tampilkan List:** Menampilkan semua data mahasiswa dalam linked list.
4. **Hitung List:** Menghitung jumlah node dalam list.
5. **Clear List:** Menghapus seluruh node dari list.

Contoh operasi: menambah, menampilkan, dan menghapus node dari linked list.

B. GUIDED II

Code :

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

// Definisi struktur untuk elemen list
struct Node
{
    int data; // Menyimpan nilai elemen
    Node *next; // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node *alokasi(int value)
{
    Node *newNode = new Node; // Alokasi memori untuk elemen baru
    if (newNode != nullptr)
    {
        // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = nullptr; // Set next ke nullptr
    }
    return newNode; // Mengembalikan pointer node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node *node)
{
    delete node; // Mengembalikan memori yang digunakan oleh node
}

// Pengecekan apakah list kosong
bool isEmpty(Node *head)
{
    return head == nullptr; // List kosong jika head adalah nullptr
}

// Menambahkan elemen di awal list
void insertFirst(Node *&head, int value)
{
    Node *newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr)
    {
        newNode->next = head; // Menghubungkan elemen baru ke elemen pertama
        head = newNode; // Menetapkan elemen baru sebagai elemen pertama
    }
}

// Menambahkan elemen di akhir list
void insertLast(Node *&head, int value)
{
    Node *newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr)
    {
        if (isEmpty(head))
        {
            // Jika list kosong
            head = newNode; // Elemen baru menjadi elemen pertama
        }
        else
        {
            Node *temp = head;
            while (temp->next != nullptr)
            { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node *head)
{
    if (isEmpty(head))
    {
        cout << "List kosong!" << endl;
    }
    else
    {
        Node *temp = head;
        while (temp != nullptr)
        {
            // Selama belum mencapai akhir list
            // Menampilkan data elemen
            // Melanjutkan ke elemen berikutnya
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
}
```

```

// Menghitung jumlah elemen dalam list
int countElements(Node *head)
{
    int count = 0;
    Node *temp = head;
    while (temp != nullptr)
    {
        count++; // Menambah jumlah elemen
        temp = temp->next; // Melanjutkan ke elemen berikutnya
    }
    return count; // Mengembalikan jumlah elemen
}

// Menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node *head)
{
    while (head != nullptr)
    {
        Node *temp = head; // Simpan pointer ke node saat ini
        head = head->next; // Pindahkan ke node berikutnya
        dealokasi(temp); // Dealokasi node
    }
}

int main()
{
    Node *head = nullptr; // Membuat list kosong

    // Menambahkan elemen ke dalam list
    insertFirst(head, 10); // Menambahkan elemen 10 di awal list
    insertLast(head, 20); // Menambahkan elemen 20 di akhir list
    insertLast(head, 30); // Menambahkan elemen 30 di akhir list

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

    // Menampilkan jumlah elemen
    cout << "Jumlah elemen: " << countElements(head) << endl;

    // Menghapus semua elemen dalam list
    clearList(head);

    // Menampilkan isi list setelah penghapusan
    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}

```

Screenshoot Output :

```

Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS C:\STD_Zivana_Afra_Yulianto>

```

Deskripsi :

Program C++ ini mengimplementasikan **Single Linked List** dengan operasi dasar seperti:

1. **Insert First:** Menambahkan elemen baru di awal list.
2. **Insert Last:** Menambahkan elemen baru di akhir list.
3. **Print List:** Menampilkan seluruh elemen dalam list.
4. **Count Elements:** Menghitung jumlah elemen dalam list.
5. **Clear List:** Menghapus seluruh elemen dan mengosongkan list.

Program mengelola memori secara manual menggunakan alokasi dan dealokasi node, serta memeriksa apakah list kosong sebelum melakukan operasi.

4. Unguided

A. UNGUIDED I

Code :

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

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

// Head pointer to the linked list
Node *head = nullptr;

// Function to insert a node at the front
void insertDepan(int nilai)
{
    Node *newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

// Function to insert a node at the back
void insertBelakang(int nilai)
{
    Node *newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

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

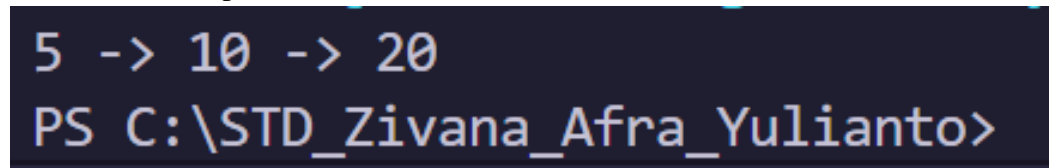
// Function to print the entire linked list
void cetakLinkedList()
{
    Node *temp = head;
    if (temp == nullptr)
    {
        cout << "Linked list kosong" << endl;
    }
    else
    {
        while (temp != nullptr)
        {
            cout << temp->data;
            if (temp->next != nullptr)
            {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
}

int main()
{
    // Contoh operasi
    insertDepan(10); // Tambah node di depan (nilai: 10)
    insertBelakang(20); // Tambah node di belakang (nilai: 20)
    insertDepan(5); // Tambah node di depan (nilai: 5)

    // Cetak isi linked list
    cetakLinkedList(); // Output: 5 -> 10 -> 20

    return 0;
}
```


Screenshoot Output :



```
5 -> 10 -> 20
PS C:\STD_Zivana_Afra_Yulianto>
```

Deskripsi :

Kode ini mengimplementasikan sebuah linked list sederhana menggunakan bahasa pemrograman C++. Terdapat beberapa komponen utama dalam kode ini:

1. **Struktur Node:**

- Struktur Node mendefinisikan elemen dari linked list, yang memiliki dua atribut: data (tipe int) dan next (pointer ke Node berikutnya).

2. **Pointer Head:**

- Pointer head digunakan untuk menunjukkan node pertama dalam linked list. Pada awalnya, head diinisialisasi dengan nullptr, menandakan bahwa linked list kosong.

3. **Fungsi insertDepan(int nilai):**

- Fungsi ini digunakan untuk menyisipkan node baru di depan linked list. Node baru diinisialisasi dengan nilai yang diberikan dan dihubungkan ke head, lalu head diperbarui untuk menunjuk ke node baru tersebut.

4. **Fungsi insertBelakang(int nilai):**

- Fungsi ini digunakan untuk menambahkan node baru di belakang linked list. Jika linked list kosong, node baru menjadi head. Jika tidak, fungsi ini akan menjelajahi linked list hingga menemukan node terakhir, kemudian menghubungkan node baru ke node tersebut.

5. **Fungsi cetakLinkedList():**

- Fungsi ini mencetak seluruh isi linked list. Jika linked list kosong, akan ditampilkan pesan "Linked list kosong". Jika tidak, akan ditampilkan nilai-nilai dalam linked list dengan format yang sesuai.

6. **Fungsi main():**

- Di dalam fungsi main, terdapat contoh operasi untuk menambahkan beberapa node (10 dan 20) dan kemudian mencetak isi linked list, yang seharusnya menghasilkan output: 5 -> 10 -> 20.

B. UNGUIDED II

Code :

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

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

// Head pointer to the linked list
Node *head = nullptr;

// Function to insert a node at the front
void insertDepan(int nilai)
{
    Node *newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

// Function to insert a node at the back
void insertBelakang(int nilai)
{
    Node *newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

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

// Function to delete a node with a specific value
void hapusNode(int nilai)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong, tidak ada node untuk dihapus" << endl;
        return;
    }

    // Jika node yang harus dihapus adalah head
    if (head->data == nilai)
    {
        Node *temp = head;
        head = head->next;
        delete temp;
        cout << "Node dengan nilai " << nilai << " berhasil dihapus" << endl;
        return;
    }

    // Mencari node dengan nilai tertentu
    Node *temp = head;
    Node *prev = nullptr;
    while (temp != nullptr && temp->data != nilai)
    {
        prev = temp;
        temp = temp->next;
    }

    // Jika node tidak ditemukan
    if (temp == nullptr)
    {
        cout << "Node dengan nilai " << nilai << " tidak ditemukan" << endl;
        return;
    }

    // Hapus node
    prev->next = temp->next;
    delete temp;
    cout << "Node dengan nilai " << nilai << " berhasil dihapus" << endl;
}
```

```

// Function to print the entire linked list
void cetakLinkedList()
{
    Node *temp = head;
    if (temp == nullptr)
    {
        cout << "Linked list kosong" << endl;
    }
    else
    {
        while (temp != nullptr)
        {
            cout << temp->data;
            if (temp->next != nullptr)
            {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
}

int main()
{
    // Contoh operasi
    insertDepan(10); // Tambah node di depan (nilai: 10)
    insertBelakang(20); // Tambah node di belakang (nilai: 20)
    insertDepan(5); // Tambah node di depan (nilai: 5)

    // Cetak isi linked list sebelum penghapusan
    cetakLinkedList(); // Output: 5 -> 10 -> 20

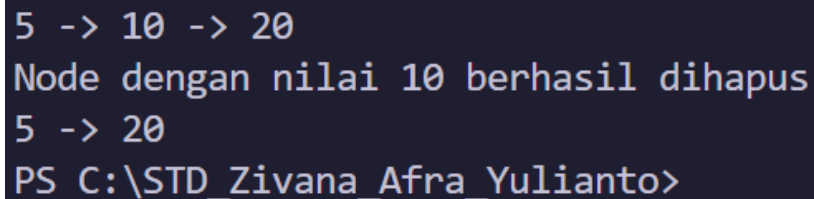
    // Hapus node dengan nilai tertentu
    hapusNode(10); // Hapus node dengan nilai 10

    // Cetak isi linked list setelah penghapusan
    cetakLinkedList(); // Output: 5 -> 20

    return 0;
}

```

Screenshoot Output :



```

5 -> 10 -> 20
Node dengan nilai 10 berhasil dihapus
5 -> 20
PS C:\STD_Zivana_Afra_Yulianto>

```

Deskripsi :

1. **Struktur Node:** Definisi elemen linked list dengan atribut data (tipe int) dan next (pointer ke node berikutnya).
2. **Pointer Head:** Pointer head menunjuk ke node pertama dalam linked list, diinisialisasi dengan nullptr jika kosong.
3. **Fungsi insertDepan(int nilai):** Menambahkan node baru di depan linked list.
4. **Fungsi insertBelakang(int nilai):** Menambahkan node baru di belakang linked list.
5. **Fungsi hapusNode(int nilai):** Menghapus node dengan nilai tertentu. Jika node ditemukan, node tersebut dihapus dan pesan konfirmasi ditampilkan.
6. **Fungsi cetakLinkedList():** Mencetak isi linked list. Jika kosong, menampilkan pesan bahwa linked list kosong.
7. **Fungsi main():** Menambahkan beberapa node, mencetak isi linked list sebelum dan setelah menghapus node dengan nilai 10.

C. UNGUIDED III

Code :

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

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

// Head pointer to the linked list
Node *head = nullptr;

// Function to insert a node at the front
void insertDepan(int nilai)
{
    Node *newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

// Function to insert a node at the back
void insertBelakang(int nilai)
{
    Node *newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

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

// Function to search for a node with a specific value
bool cariNode(int nilai)
{
    Node *temp = head;
    while (temp != nullptr)
    {
        if (temp->data == nilai)
        {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

// Function to count the length of the linked list
int hitungPanjangLinkedList()
{
    Node *temp = head;
    int count = 0;
    while (temp != nullptr)
    {
        count++;
        temp = temp->next;
    }
    return count;
}
```

```

// Function to print the entire linked list
void cetakLinkedList()
{
    Node *temp = head;
    if (temp == nullptr)
    {
        cout << "Linked list kosong" << endl;
    }
    else
    {
        while (temp != nullptr)
        {
            cout << temp->data;
            if (temp->next != nullptr)
            {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
}

int main()
{
    // Contoh operasi
    insertDepan(10); // Tambah node di depan (nilai: 10)
    insertBelakang(20); // Tambah node di belakang (nilai: 20)
    insertDepan(5); // Tambah node di depan (nilai: 5)

    // Cetak isi linked list
    cetakLinkedList(); // Output: 5 -> 10 -> 20

    // Mencari node dengan nilai tertentu
    int nilaiDicari = 20;
    if (cariNode(nilaiDicari))
    {
        cout << "Node dengan nilai " << nilaiDicari << " ditemukan." << endl;
    }
    else
    {
        cout << "Node dengan nilai " << nilaiDicari << " tidak ditemukan." << endl;
    }

    // Hitung panjang linked list
    int panjang = hitungPanjangLinkedList();
    cout << "Panjang linked list: " << panjang << endl;

    return 0;
}

```

Screenshoot Output :

```

5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS C:\STD_Zivana_Afra_Yulianto>

```

Deskripsi :

1. **Struktur Node:** Menyimpan data (tipe int) dan pointer next ke node berikutnya.
2. **Pointer Head:** Menunjuk ke node pertama dalam linked list, diinisialisasi dengan nullptr jika kosong.
3. **Fungsi insertDepan(int nilai):** Menambahkan node baru di depan linked list.
4. **Fungsi insertBelakang(int nilai):** Menambahkan node baru di belakang linked list.
5. **Fungsi cariNode(int nilai):** Mencari node dengan nilai tertentu. Mengembalikan true jika ditemukan, atau false jika tidak.
6. **Fungsi hitungPanjangLinkedList():** Menghitung dan mengembalikan jumlah node dalam linked list.
7. **Fungsi cetakLinkedList():** Mencetak seluruh isi linked list. Jika kosong, menampilkan pesan bahwa linked list kosong.
8. **Fungsi main():** Menambahkan beberapa node, mencetak isi linked list, mencari node dengan nilai tertentu, dan menghitung panjang linked list.

5. Kesimpulan

Praktikum ini berhasil menunjukkan cara implementasi dan manipulasi struktur data Single Linked List dalam bahasa C++. Beberapa kesimpulan yang dapat diambil adalah:

- Single Linked List merupakan struktur data yang efisien untuk menyimpan data secara dinamis.
- Operasi dasar seperti penambahan, penghapusan, dan pencarian dapat dilakukan dengan efektif menggunakan teknik pengelolaan pointer.
- Penggunaan memori yang fleksibel dalam Linked List memungkinkan pengembang untuk mengelola data dengan lebih efisien, terutama saat ukuran data tidak dapat diprediksi.
- Praktikum ini meningkatkan pemahaman tentang pengelolaan memori dan penggunaan struktur data yang tepat dalam pemrograman.