LAPORAN PRAKTIKUM

Modul 4

Single Linked List (Bagian Pertama)

Disusun Oleh:

Yogi Hafidh Maulana - 2211104061

SE06-02

Dosen :

Wahyu Andi

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

- Memahami konsep Single Linked List.
- Mengimplementasikan operasi dasar pada Single Linked List.
- Mengelola memori dinamis dalam Single Linked List.
- Mengembangkan program berbasis Single Linked List.
- Menerapkan konsep Single Linked List dalam pemrograman praktis.

2. Landasan Teori

- Pengertian Linked List
  Linked List adalah salah satu bentuk struktur data yang menyimpan serangkaian elemen yang saling terhubung satu sama lain melalui pointer. Setiap elemen dalam linked list disebut node, yang terdiri dari dua bagian: data (informasi yang disimpan dalam node) dan pointer (penunjuk ke node berikutnya). Berbeda dengan array, linked list bersifat dinamis, sehingga ukurannya dapat berubah sesuai kebutuhan (bertambah atau berkurang).
- Single Linked List
  Single Linked List adalah salah satu varian dari linked list di mana setiap node hanya memiliki satu pointer, yaitu penunjuk ke node berikutnya (successor). Dalam single linked list, pengaksesan elemen hanya dapat dilakukan secara sekuensial (dari depan ke belakang), karena node hanya memiliki satu arah penunjuk.
- Keunggulan Linked List Dibandingkan Array
  Linked list memiliki beberapa keunggulan dibandingkan array, di antaranya:
  Ukuran Dinamis: Linked list dapat tumbuh dan mengecil sesuai kebutuhan, sedangkan array memiliki ukuran tetap.
  Efisiensi dalam Penyisipan dan Penghapusan: Pada linked list, penambahan atau penghapusan elemen pada posisi awal atau akhir bisa dilakukan dengan efisien tanpa perlu menggeser elemen-elemen lain, seperti yang harus dilakukan pada array.
  Namun, linked list juga memiliki kelemahan, seperti waktu akses yang lebih lambat dibandingkan array, karena linked list tidak menyediakan akses langsung ke elemen tertentu (harus dilakukan pencarian sekuensial).

3. Guided

a) Guided 1

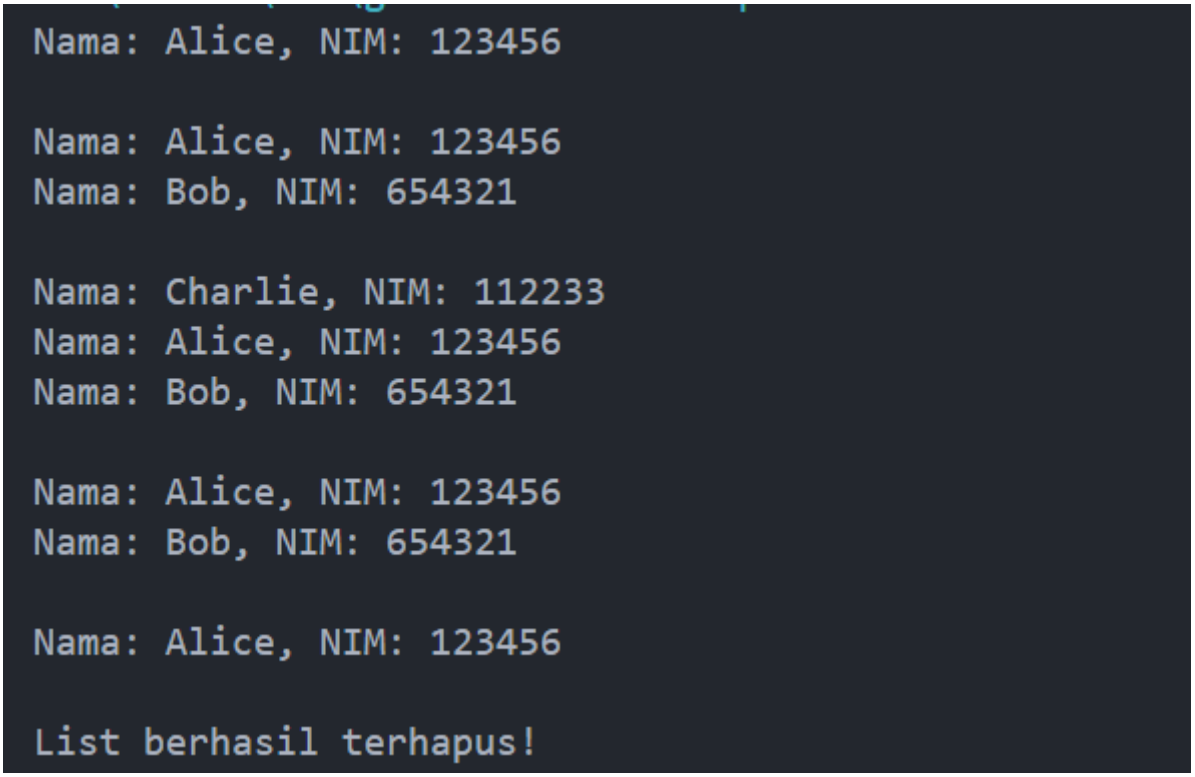
Code:

```cpp
#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;
}
```

```cpp
// 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;
    }
}
```

```cpp
// 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;
}
```

Output:



Deskripsi Program:

Program di atas adalah implementasi Single Linked List yang menyimpan data mahasiswa dengan atribut nama dan NIM. Program ini memiliki beberapa operasi dasar: inisialisasi list, pengecekan apakah list kosong, menambah node di depan (insertDepan), menambah node di belakang (insertBelakang), menghitung jumlah node di dalam list (hitungList), menghapus node di depan (hapusDepan), menghapus node di belakang (hapusBelakang), menampilkan seluruh isi list (tampil), dan menghapus seluruh node dalam list (clearList). Di bagian main, program melakukan beberapa operasi seperti menambahkan mahasiswa ke list, menampilkan isi list, serta menghapus node secara bertahap dan mengosongkan seluruh list pada akhirnya.

b) Guided 2

Code:

```cpp
#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 isListEmpty(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
    }
}
```

```cpp
// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(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 (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        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
}
```

```cpp
// 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;
}
```

Output:

```
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS D:\PROJECT\C++ Project\Laprak 4>
```

Deskripsi Program:

Program di atas adalah implementasi Single Linked List dalam C++ yang menyediakan beberapa operasi dasar seperti menambahkan elemen di awal dan di akhir list, menampilkan isi list, menghitung jumlah elemen dalam list, serta menghapus seluruh elemen dalam list dan mendekalokasi memori yang digunakan. Struktur Node menyimpan data integer dan pointer ke node berikutnya. Fungsi insertFirst menambahkan node di awal, sedangkan insertLast menambah node di akhir. Fungsi printList digunakan untuk menampilkan elemen list, dan countElements untuk menghitung jumlah elemen. Fungsi clearList menghapus seluruh elemen dalam linked list dengan melakukan dealokasi pada setiap node. Program juga mendemonstrasikan operasi-operasi ini pada list yang terdiri dari elemen-elemen 10, 20, dan 30.

4. Unguided

a) Buatlah program C++ untuk membuat sebuah single linked list dengan operasi dasar

sebagai berikut:

● Insert Node di Depan: Fungsi untuk menambah node baru di awal linked list.

● Insert Node di Belakang: Fungsi untuk menambah node baru di akhir linked list.

● Cetak Linked List: Fungsi untuk mencetak seluruh isi linked list.

Contoh input dan output:

Input:

1. Tambah node di depan (nilai: 10)

2. Tambah node di belakang (nilai: 20)

3. Tambah node di depan (nilai: 5)

4. Cetak linked list

Output:

5 -> 10 -> 20

Code:

```cpp
#include <iostream>

using namespace std;

// Definisi struktur node
struct Node
{
    int data;
    Node *next;
};

// Fungsi untuk membuat node baru
Node *createNode(int value)
{
    Node *newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Fungsi untuk menambah node di depan
void insertAtFront(Node *&head, int value)
{
    Node *newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menambah node di belakang
void insertAtEnd(Node *&head, int value)
{
    Node *newNode = createNode(value);
    if (head == nullptr)
    {
        head = newNode; // Jika linked list kosong
    }
    else
    {
        Node *temp = head;
        while (temp->next != nullptr)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```cpp
// Fungsi untuk mencetak seluruh isi linked list
void printList(Node *head)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong." << endl;
        return;
    }

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

// Fungsi utama
int main()
{
    Node *head = nullptr; // Inisialisasi linked list kosong

    // Contoh penggunaan fungsi
    insertAtFront(head, 10); // Tambah node di depan (10)
    insertAtEnd(head, 20);   // Tambah node di belakang (20)
    insertAtFront(head, 5);  // Tambah node di depan (5)

    // Cetak linked list
    cout << "Isi linked list: ";
    printList(head);

    return 0;
}
```

Output:

```
Isi linked list: 5 -> 10 -> 20
PS D:\PROJECT\C++ Project\Laprak 4>
```

b) Buatlah program C++ yang dapat menghapus node tertentu dalam single linked list
berdasarkan nilai yang diberikan oleh pengguna. Tugas ini mencakup operasi:
● Delete Node dengan Nilai Tertentu: Fungsi untuk menghapus node yang memiliki nilai tertentu.
● Cetak Linked List: Setelah penghapusan, cetak kembali isi linked list.
Contoh input/output:
Input:
1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Hapus node dengan nilai (nilai: 10)
5. Cetak linked list
Output:
5 -> 20

Code:

```cpp
#include <iostream>

using namespace std;

// Definisi struktur node
struct Node
{
    int data;
    Node *next;
};

// Fungsi untuk membuat node baru
Node *createNode(int value)
{
    Node *newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Fungsi untuk menambah node di depan
void insertAtFront(Node *&head, int value)
{
    Node *newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menambah node di belakang
void insertAtEnd(Node *&head, int value)
{
    Node *newNode = createNode(value);
    if (head == nullptr)
    {
        head = newNode; // Jika linked list kosong
    }
    else
    {
        Node *temp = head;
        while (temp->next != nullptr)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```cpp
// Fungsi untuk menghapus node dengan nilai tertentu
void deleteNode(Node *&head, int value)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong, tidak ada yang dapat dihapus." << endl;
        return;
    }

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

    // Mencari node yang memiliki nilai tersebut
    Node *temp = head;
    while (temp->next != nullptr && temp->next->data != value)
    {
        temp = temp->next;
    }

    // Jika node ditemukan
    if (temp->next != nullptr)
    {
        Node *nodeToDelete = temp->next;
        temp->next = nodeToDelete->next;
        delete nodeToDelete;
        cout << "Node dengan nilai " << value << " berhasil dihapus." << endl;
    }
    else
    {
        cout << "Node dengan nilai " << value << " tidak ditemukan." << endl;
    }
}
```

```cpp
// Fungsi untuk mencetak seluruh isi linked list
void printList(Node *head)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong." << endl;
        return;
    }

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

// Fungsi utama
int main()
{
    Node *head = nullptr; // Inisialisasi linked list kosong

    // Contoh penggunaan fungsi
    insertAtFront(head, 10); // Tambah node di depan (10)
    insertAtEnd(head, 20);   // Tambah node di belakang (20)
    insertAtFront(head, 5);  // Tambah node di depan (5)

    cout << "Isi linked list sebelum penghapusan: ";
    printList(head);

    deleteNode(head, 10); // Hapus node dengan nilai 10

    cout << "Isi linked list setelah penghapusan: ";
    printList(head);

    return 0;
}
```

Output:

```
Isi linked list sebelum penghapusan: 5 -> 10 -> 20
Node dengan nilai 10 berhasil dihapus.
Isi linked list setelah penghapusan: 5 -> 20
PS D:\PROJECT\C++ Project\Laprak 4>
```

c) Buatlah program C++ yang dapat melakukan operasi berikut:

● Cari Node dengan Nilai Tertentu: Fungsi untuk mencari apakah sebuah nilai ada di dalam linked list.

● Hitung Panjang Linked List: Fungsi untuk menghitung jumlah node yang ada di dalam linked list.

Contoh input/output:

Input:

1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cari node dengan nilai 20
5. Cetak panjang linked list

Output:

Node dengan nilai 20 ditemukan.

Panjang linked list: 3

Code:

```cpp
#include <iostream>

using namespace std;

// Definisi struktur node
struct Node
{
    int data;
    Node *next;
};

// Fungsi untuk membuat node baru
Node *createNode(int value)
{
    Node *newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Fungsi untuk menambah node di depan
void insertAtFront(Node *&head, int value)
{
    Node *newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menambah node di belakang
void insertAtEnd(Node *&head, int value)
{
    Node *newNode = createNode(value);
    if (head == nullptr)
    {
        head = newNode; // Jika linked list kosong
    }
    else
    {
        Node *temp = head;
        while (temp->next != nullptr)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```cpp
// Fungsi untuk mencari node dengan nilai tertentu
bool searchNode(Node *head, int value)
{
    Node *temp = head;
    while (temp != nullptr)
    {
        if (temp->data == value)
        {
            return true; // Nilai ditemukan
        }
        temp = temp->next;
    }
    return false; // Nilai tidak ditemukan
}

// Fungsi untuk menghitung panjang linked list
int getLength(Node *head)
{
    int length = 0;
    Node *temp = head;
    while (temp != nullptr)
    {
        length++;
        temp = temp->next;
    }
    return length;
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node *head)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong." << endl;
        return;
    }

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

```cpp
// Fungsi utama
int main()
{
    Node *head = nullptr; // Inisialisasi linked list kosong

    // Contoh penggunaan fungsi
    insertAtFront(head, 10); // Tambah node di depan (10)
    insertAtEnd(head, 20);   // Tambah node di belakang (20)
    insertAtFront(head, 5);  // Tambah node di depan (5)

    // Cetak seluruh isi linked list
    cout << "Isi linked list: ";
    printList(head);

    // Cari node dengan nilai 20
    int searchValue = 20;
    if (searchNode(head, searchValue))
    {
        cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
    }
    else
    {
        cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
    }

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

    return 0;
}
```

Output:

```
Isi linked list: 5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS D:\PROJECT\C++ Project\Laprak 4>
```

5. Kesimpulan

Single Linked List adalah struktur data dinamis yang terdiri dari serangkaian elemen (node) yang saling terhubung melalui pointer, di mana setiap node berisi data dan penunjuk ke elemen berikutnya. Struktur ini menawarkan keunggulan dalam hal fleksibilitas ukuran, karena dapat bertambah atau berkurang sesuai kebutuhan, serta efisiensi dalam penyisipan dan penghapusan elemen tanpa perlu menggeser elemen lainnya. Operasi-operasi dasar pada Single Linked List meliputi inisialisasi, penyisipan di awal dan akhir, penghapusan elemen, pencarian, dan penelusuran isi list. Dalam implementasinya, Single Linked List memanfaatkan manajemen memori dinamis menggunakan alokasi dan dealokasi memori dengan operator pointer, sehingga penggunaan memori menjadi lebih efisien. Dengan demikian, Single Linked List merupakan struktur data yang penting untuk dipahami dalam pemrograman, khususnya dalam kasus yang membutuhkan manipulasi data secara fleksibel.