LAPORAN PRAKTIKUM
Modul 4
SINGLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:
Aulia Jasifa Br Ginting 2311104060
S1SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **Tujuan**
    1. Memahami penggunaan *linked list* dengan *pointer* operator-operator dalam program.
    2. Memahami operasi-operasi dasar *linked list*.
    3. Membuat program dengan menggunakan *linkend list* dengan *prototype* yang ada.
2. **Landasan Teori**

4.1 Linked List dengan Pointer

Linked list dengan pointer adalah sebuah struktur data di mana elemen-elemennya dihubungkan satu sama lain menggunakan pointer. Setiap elemen dalam linked list disebut **node**, dan setiap node berisi dua bagian utama:

1. **Data (Value)**: Bagian yang menyimpan informasi atau nilai dari elemen tersebut.
2. **Pointer (Next)**: Bagian yang menyimpan alamat atau referensi ke node berikutnya dalam urutan.

Dengan menggunakan pointer, setiap node dalam linked list dapat "menunjuk" ke node selanjutnya, sehingga menciptakan struktur data yang dinamis dan fleksibel. Berikut adalah beberapa karakteristik penting dari linked list dengan pointer:

4.2 Single Linked List

Single Linked List adalah salah satu jenis linked list di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya dalam daftar. Struktur ini bersifat linear, yang berarti Anda dapat menelusuri linked list hanya dari satu arah, yaitu dari head (node pertama) ke tail (node terakhir), sampai menemukan null atau akhir dari daftar.

4.2.1 Single Linked List

A. Pembentukan List

Adalah proses pembuatan dan penambahan elemen-elemen (node) ke dalam linked list dari awal hingga membentuk struktur lengkap. Pembentukan linked list melibatkan pengaturan node pertama (head) dan penambahan node-node lainnya secara berurutan menggunakan pointer.

B. Pengalokasian Memori

Adalah proses reservasi atau penyediaan ruang memori yang diperlukan oleh suatu program atau data selama eksekusi program. Dalam pemrograman, pengalokasian memori digunakan untuk menyimpan variabel, objek, struktur data (seperti array dan linked list), dan elemen-elemen lainnya sehingga program dapat berjalan dengan benar.

C. **Dealokasi**

Adalah proses mengembalikan atau membebaskan kembali ruang memori yang sebelumnya telah dialokasikan untuk program atau struktur data, sehingga memori tersebut dapat digunakan lagi oleh sistem. Dalam bahasa pemrograman seperti C dan C++, dealokasi sangat penting untuk mencegah kebocoran memori (**memory leak**) yang terjadi ketika memori yang dialokasikan tidak pernah dibebaskan, menyebabkan sistem kehabisan memori yang tersedia.

D. **Pengecekan List**

Aadalah proses untuk memeriksa atau menentukan kondisi tertentu dari linked list, seperti apakah list kosong, apakah suatu elemen terdapat dalam list, atau berapa banyak elemen yang ada dalam list. Pengecekan ini penting untuk memastikan integritas dan validitas operasi yang akan dilakukan pada linked list.

4.2.2 Insert

A. **Insert First**

Insert First pada **singly linked list** adalah operasi untuk menambahkan sebuah node baru di **awal** linked list. Ini berarti node baru akan menjadi node pertama (head) dari linked list, dan node sebelumnya yang menjadi head akan bergeser ke posisi kedua.

B. **Insert Last**

Insert Last pada **singly linked list** adalah operasi untuk menambahkan sebuah node baru di **akhir** linked list. Setelah operasi ini, node baru menjadi node terakhir dalam daftar, dan pointer next dari node sebelumnya yang tadinya merupakan node terakhir akan menunjuk ke node baru. Pointer next dari node baru akan diatur ke null, karena itu adalah akhir dari linked list.

C. **Insert Last**

Insert After pada **singly linked list** adalah operasi untuk menambahkan sebuah node baru **setelah node tertentu** yang sudah ada dalam linked list. Dengan operasi ini, node baru akan dimasukkan tepat setelah node yang ditentukan, dan node baru akan menjadi penerus node tersebut.

4.2.3 View

Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut.

3. Guided

1. Guided 1

   Programnya

```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;
    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;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMES
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!
```

2. Guided 2

Programnya

```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
    }

}

// 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
}

// 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
    }
    cout << endl;
}

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;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMESTER 3\
Isi List: 10 20 30
Jumlah elemen: 3

Isi List setelah penghapusan: List kosong!
```

4. **Unguided**
    1. **Membuat Single Linked List**

       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.

Telkom University Purwokerto

Programnya

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

// Struktur untuk node dalam linked list
struct Node {
    int data;
    Node* next;
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Fungsi untuk menambah node di depan
    void insertFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    // Fungsi untuk menambah node di belakang
    void insertBack(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

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

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

    // Fungsi untuk mencetak linked list
    void printList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList list;

    int choice, value;
    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah node di depan\n";
        cout << "2. Tambah node di belakang\n";
        cout << "3. Cetak linked list\n";
        cout << "4. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan nilai: ";
                cin >> value;
                list.insertFront(value);
                break;
            case 2:
                cout << "Masukkan nilai: ";
                cin >> value;
                list.insertBack(value);
                break;
            case 3:
                cout << "Linked List: ";
                list.printList();
                break;
            case 4:
                cout << "Program selesai.\n";
                break;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (choice != 4);

    return 0;
}
```

Outputnya:

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 1
Masukkan nilai: 10

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 2
Masukkan nilai: 20

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 1
Masukkan nilai: 5

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 3
Linked List: 5 -> 10 -> 20
```

2. **Menghapus Node pada Linked List**
   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.

Telkom University Purwokerto

Programnya

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

// Struktur untuk node dalam linked list
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Fungsi untuk menambah node di depan
    void tambahDepan(int nilai) {
        Node* newNode = new Node(nilai);
        newNode->next = head;
        head = newNode;
    }

    // Fungsi untuk menambah node di belakang
    void tambahBelakang(int nilai) {
        Node* newNode = new Node(nilai);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    // Fungsi untuk menghapus node dengan nilai tertentu
    void hapusNode(int nilai) {
        if (!head) return;

        if (head->data == nilai) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Node* current = head;
        Node* prev = nullptr;
        while (current && current->data != nilai) {
            prev = current;
            current = current->next;
        }

        if (!current) return;

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

    // Fungsi untuk mencetak linked list
    void cetakList() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << " -> ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList list;
    int pilihan, nilai;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah node di depan\n";
        cout << "2. Tambah node di belakang\n";
        cout << "3. Hapus node dengan nilai tertentu\n";
        cout << "4. Cetak linked list\n";
        cout << "5. Keluar\n";
        cout << "Pilih operasi: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan nilai: ";
                cin >> nilai;
                list.tambahDepan(nilai);
                break;
            case 2:
                cout << "Masukkan nilai: ";
                cin >> nilai;
                list.tambahBelakang(nilai);
                break;
            case 3:
                cout << "Masukkan nilai yang akan dihapus: ";
                cin >> nilai;
                list.hapusNode(nilai);
                break;
            case 4:
                cout << "Isi linked list: ";
                list.cetakList();
                break;
            case 5:
                cout << "Program selesai.\n";
                return 0;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    }

    return 0;
}
```

Outputnya:

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 1
Masukkan nilai: 10

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 2
Masukkan nilai: 20

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 1
Masukkan nilai: 5
```

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 3
Masukkan nilai yang akan dihapus: 10

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 4
Isi linked list: 5 -> 20
```

3. **Mencari dan Menghitung Panjang Linked List**

   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.

Telkom University Purwokerto

Programnya

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

// Struktur untuk node linked list
struct Node {
    int data;
    Node* next;

    // Constructor
    Node(int value) {
        data = value;
        next = NULL;
    }
};

class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = NULL;
    }

    // Fungsi untuk menambah node di depan
    void tambahDepan(int nilai) {
        Node* newNode = new Node(nilai);
        newNode->next = head;
        head = newNode;
        cout << "Node dengan nilai " << nilai << " ditambahkan di depan.\n";
    }

    // Fungsi untuk menambah node di belakang
    void tambahBelakang(int nilai) {
        Node* newNode = new Node(nilai);

        if (head == NULL) {
            head = newNode;
            cout << "Node dengan nilai " << nilai << " ditambahkan di belakang.\n";
            return;
        }

        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        cout << "Node dengan nilai " << nilai << " ditambahkan di belakang.\n";
    }

    // Fungsi untuk mencari node dengan nilai tertentu
    bool cariNode(int nilai) {
        Node* current = head;
        while (current != NULL) {
            if (current->data == nilai) {
                cout << "Node dengan nilai " << nilai << " ditemukan.\n";
                return true;
            }
            current = current->next;
        }
        cout << "Node dengan nilai " << nilai << " tidak ditemukan.\n";
        return false;
    }

    // Fungsi untuk menghitung panjang linked list
    int hitungPanjang() {
        int count = 0;
        Node* current = head;
        while (current != NULL) {
            count++;
            current = current->next;
        }
        cout << "Panjang linked list: " << count << endl;
        return count;
    }
};

int main() {
    LinkedList list;
    int pilihan, nilai;

    do {
        cout << "\nMenu Operasi Linked List:\n";
        cout << "1. Tambah node di depan\n";
        cout << "2. Tambah node di belakang\n";
        cout << "3. Cari node\n";
        cout << "4. Hitung panjang linked list\n";
        cout << "5. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan nilai: ";
                cin >> nilai;
                list.tambahDepan(nilai);
                break;

            case 2:
                cout << "Masukkan nilai: ";
                cin >> nilai;
                list.tambahBelakang(nilai);
                break;

            case 3:
                cout << "Masukkan nilai yang dicari: ";
                cin >> nilai;
                list.cariNode(nilai);
                break;

            case 4:
                list.hitungPanjang();
                break;

            case 5:
                cout << "Program selesai.\n";
                break;

            default:
                cout << "Pilihan tidak valid!\n";
        }
    } while (pilihan != 5);

    return 0;
}
```

Outputnya:

```
Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 1
Masukkan nilai: 10
Node dengan nilai 10 ditambahkan di depan.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 2
Masukkan nilai: 20
Node dengan nilai 20 ditambahkan di belakang.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 1
Masukkan nilai: 5
Node dengan nilai 5 ditambahkan di depan.
```

```
Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 3
Masukkan nilai yang dicari: 20
Node dengan nilai 20 ditemukan.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 4
Panjang linked list: 3
```

5. Kesimpulan

Pada materi ini saya dapat memahami dan cara penggunaan Linked List struktur data yang sangat berguna, terutama ketika kita tidak tahu sebelumnya berapa banyak data yang akan disimpan atau ketika kita sering melakukan penambahan dan penghapusan elemen di tengah data. Serta dapat memahami mengenai kelebihan dan kekurangannya sebelum memilih struktur data dalam suatu aplikasi.