# LAPORAN PRAKTIKUM

# PERTEMUAN 4


**Disusun Oleh:**

**Naura Aisha Zahira (2311104078)**

**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 dalam linked list.
3) Membuat program dengan menggunakan linked list dengan prototype yang ada.

## 2. Landasan Teori

### 1) Linked List dengan Pointer

Linked list adalah struktur data yang terdiri dari elemen-elemen (disebut *node*) yang saling terhubung satu sama lain menggunakan pointer. Setiap node dalam linked list menyimpan dua komponen utama:

a. Data: Informasi yang akan disimpan dalam node, bisa berupa angka, string, atau tipe data lainnya.
b. Pointer: Penunjuk ke node berikutnya dalam urutan.

Keunggulan utama linked list dibandingkan struktur data lain seperti array adalah fleksibilitas dalam ukuran. Node dapat dengan mudah ditambah atau dihapus tanpa harus mengalokasikan ulang blok memori secara berurutan. Pointer bertugas menjaga keterhubungan antar node, memungkinkan operasi seperti penambahan dan penghapusan data di mana saja dalam list tanpa memindahkan elemen lainnya.

Operasi dasar pada linked list melibatkan pengelolaan pointer, seperti:

- Insert: Menyisipkan node baru ke dalam list, baik di awal, di tengah, atau di akhir.
- Delete: Menghapus node tertentu dari list.
- Traversal: Mengunjungi setiap node dalam list secara berurutan.
- Search: Mencari node berdasarkan nilai data yang disimpan.

Pointer memungkinkan linked list memiliki alokasi memori yang dinamis, di mana node dialokasikan dan dibebaskan secara fleksibel saat program berjalan.

### 2) Single Linked List

Single linked list adalah tipe linked list yang paling dasar. Setiap node dalam single linked list hanya memiliki satu pointer, yaitu pointer ke node berikutnya (next pointer). Dalam struktur ini, hubungan antar node bersifat searah, dimulai dari node pertama yang disebut head hingga node terakhir, di mana pointer pada node terakhir akan menunjuk ke null atau nullptr (penanda akhir list).

Operasi-operasi utama dalam single linked list meliputi:

a. Inisialisasi List: Proses mempersiapkan linked list baru dengan mengosongkan head (mengarahkannya ke null).

b. Penambahan Node:
    - Insert Depan (*Insert First*): Menyisipkan node baru di awal list dengan membuat node baru menjadi head dan menunjuk ke node sebelumnya.
    - Insert Belakang (*Insert Last*): Menyisipkan node baru di akhir list dengan mengarahkan pointer node terakhir sebelumnya ke node baru.

c. Penghapusan Node:
    - Delete Depan (*Delete First*): Menghapus node pertama (head) dengan memindahkan head ke node berikutnya.
    - Delete Belakang (*Delete Last*): Menghapus node terakhir dengan mencari node yang penunjuk berikutnya adalah null, kemudian memutus hubungan ke node terakhir.

d. Traversal: Mengunjungi setiap node dalam linked list secara berurutan dari head ke node terakhir untuk memeriksa atau menampilkan data.

e. Penghitungan Elemen: Menghitung jumlah node dengan traversal dari head hingga akhir.

Karakteristik utama single linked list adalah efisiensi dalam penambahan dan penghapusan elemen dari awal atau akhir list. Namun, untuk operasi yang terjadi di tengah-tengah list, traversal dari head ke posisi yang diinginkan diperlukan, yang bisa memperlambat proses jika list memiliki banyak elemen.

## 3. Guided

1) Code:

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

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

// Deklaraci 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;
    }
}

// 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(m1);
    tampil();

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

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

Penjelasan code:

a. Struktur Data:
   - **mahasiswa**: Struktur ini mendefinisikan data yang akan disimpan dalam setiap node, yaitu nama dan NIM mahasiswa.
   - **Node**: Struktur ini mewakili setiap node dalam linked list. Setiap node memiliki data mahasiswa dan pointer *next* yang menunjuk ke node berikutnya.

b. Variabel Global:
   - *head*: Pointer ke node pertama dalam list.
   - *tail*: Pointer ke node terakhir dalam list.

c. Fungsi-fungsi Utama:
   - **init():** Inisialisasi list dengan membuat *head* dan *tail* menunjuk ke *nullptr* (kosong).
   - **isEmpty():** Mengecek apakah list kosong.
   - **insertDepan() dan insertBelakang():** Menambahkan node baru ke depan atau belakang list.
   - **hitungList():** Menghitung jumlah node dalam list.
   - **hapusDepan() dan hapusBelakang():** Menghapus node pertama atau terakhir dari list.
   - **tampil():** Menampilkan semua data mahasiswa dalam list.
   - **clearList()**: Menghapus semua node dalam list.

d. Cara Kerja:
   - Program dimulai dengan menginisialisasi list.
   - Kemudian, beberapa data mahasiswa ditambahkan ke dalam list menggunakan fungsi *insertDepan()* dan *insertBelakang()*.
   - Setelah itu, beberapa elemen dihapus menggunakan fungsi *hapusDepan()* dan *hapusBelakang()*.
   - Terakhir, seluruh list dihapus menggunakan fungsi *clearList()*.

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!
```

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
    }
}

// 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
    }
}

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;
}
```

Penjelasan Code:

a. Struktur Node:
   - *data*: Menyimpan nilai data yang ingin disimpan dalam node.
   - *next*: Pointer yang menunjuk ke node berikutnya dalam list.

b. Fungsi-fungsi Utama:
   - *alokasi()*: Mengalokasikan memori untuk node baru dan menginisialisasi data dan pointernya.
   - *dealokasi()*: Membebaskan memori yang digunakan oleh sebuah node.
   - *isListEmpty()*: Mengecek apakah list kosong.
   - *insertFirst()*: Menambahkan node baru di awal list.
   - *insertLast()*: Menambahkan node baru di akhir list.
   - *printList()*: Menampilkan semua data dalam list.
   - *countElements()*: Menghitung jumlah elemen dalam list.
   - *clearList()*: Menghapus semua elemen dalam list dan membebaskan memori.

c. Cara Kerja:
   - Program dimulai dengan membuat list kosong.
   - Kemudian, elemen-elemen ditambahkan ke dalam list menggunakan fungsi *insertFirst()* dan *insertLast()*.
   - Setelah itu, isi list ditampilkan menggunakan fungsi *printList()*.
   - Jumlah elemen dalam list dihitung menggunakan fungsi *countElements()*.
   - Terakhir, semua elemen dihapus dari list menggunakan fungsi *clearList()*.

## 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.

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;

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

// Deklarasi pointer untuk head dan tail
Node* head = nullptr;
Node* tail = nullptr;

// Fungsi untuk menambah node di depan
void insertDepan(int nilai) {
    Node* baru = new Node;
    baru->data = nilai;
    baru->next = nullptr;

    if (head == nullptr) {
        // Jika list kosong, node pertama menjadi head dan tail
        head = tail = baru;
    } else {
        // Jika list tidak kosong, node baru menjadi head
        baru->next = head;
        head = baru;
    }
}

// Fungsi untuk menambah node di belakang
void insertBelakang(int nilai) {
    Node* baru = new Node;
    baru->data = nilai;
    baru->next = nullptr;

    if (head == nullptr) {
        // Jika list kosong, node pertama menjadi head dan tail
        head = tail = baru;
    } else {
        // Jika list tidak kosong, node baru ditambahkan di belakang tail
        tail->next = baru;
        tail = baru;
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList() {
    if (head == nullptr) {
        cout << "Linked list kosong.\n";
    } else {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << " -> ";
            }
            current = current->next;
        }
        cout << endl;
    }
}

// Main function
int main() {
    // Contoh penggunaan fungsi
    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
    cetakList();

    return 0;
}

```

Output:

```
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.

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;

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

// Deklarasi pointer untuk head dan tail
Node* head = nullptr;
Node* tail = nullptr;

// Fungsi untuk menambah node di depan
void insertDepan(int nilai) {
    Node* baru = new Node;
    baru->data = nilai;
    baru->next = nullptr;

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

// Fungsi untuk menambah node di belakang
void insertBelakang(int nilai) {
    Node* baru = new Node;
    baru->data = nilai;
    baru->next = nullptr;

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

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

    // Jika node yang dihapus adalah head
    if (head->data == nilai) {
        Node* hapus = head;
        head = head->next;
        delete hapus;
        return;
    }

    // Mencari node yang akan dihapus
    Node* current = head;
    Node* prev = nullptr;
    while (current != nullptr && current->data != nilai) {
        prev = current;
        current = current->next;
    }

    // Jika node ditemukan
    if (current != nullptr) {
        prev->next = current->next;
        // Jika node yang dihapus adalah tail, perbarui tail
        if (current == tail) {
            tail = prev;
        }
        delete current;
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList() {
    if (head == nullptr) {
        cout << "Linked list kosong.\n";
    } else {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << "->";
            }
            current = current->next;
        }
        cout << endl;
    }
}

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

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

    // Cetak isi linked list setelah penghapusan
    cetakList();

    return 0;
}

```

Output:

```
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.

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;

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

// Fungsi untuk menambahkan node di depan
Node *insertAtBeginning(Node *head, int data) {
  Node *newNode = new Node;
  newNode->data = data;
  newNode->next = head;
  return newNode;
}

// Fungsi untuk menambahkan node di belakang
Node *insertAtEnd(Node *head, int data) {
  Node *newNode = new Node;
  newNode->data = data;
  newNode->next = NULL;
  if (head == NULL) {
    return newNode;
  }
  Node *temp = head;
  while (temp->next != NULL) {
    temp = temp->next;
  }
  temp->next = newNode;
  return head;
}

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

// Fungsi untuk menghitung panjang linked list
int countNodes(Node *head) {
  if (head == NULL) {
    return 0;
  }
  Node *temp = head;
  int count = 0;
  while (temp != NULL) {
    count++;
    temp = temp->next;
  }
  return count;
}

int main() {
  Node *head = NULL;
  head = insertAtBeginning(head, 10);
  head = insertAtEnd(head, 20);
  head = insertAtBeginning(head, 5);

  if (searchNode(head, 20)) {
    cout << "Node dengan nilai 20 ditemukan." << endl;
  } else {
    cout << "Node dengan nilai 20 tidak ditemukan." << endl;
  }

  cout << "Panjang linked list: " << countNodes(head) << endl;

  return 0;
}
```

Output:

```
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```

## 5. Kesimpulan

Pada praktikum ini, penggunaan linked list telah berhasil diimplementasikan dengan berbagai operasi dasar seperti penambahan, penghapusan, dan penampilan node. Dari hasil yang diperoleh, dapat disimpulkan bahwa:

1) Single linked list memudahkan dalam penambahan dan penghapusan elemen tanpa perlu menggeser data lainnya seperti dalam array.
2) Dengan memanfaatkan pointer, linked list memberikan fleksibilitas dalam pengelolaan memori, karena elemen-elemen dapat dialokasikan dan dibebaskan secara dinamis.
3) Operasi-operasi dasar seperti insert, delete, dan print list berjalan sesuai dengan yang diharapkan, dan penghitungan jumlah node juga berhasil dilakukan dengan akurat.
4) Pemahaman tentang pointer dan struktur data sangat penting dalam penggunaan linked list, karena kesalahan dalam mengelola pointer dapat menyebabkan kebocoran memori atau akses memori yang tidak valid.

Praktikum ini memberikan pemahaman yang lebih mendalam tentang bagaimana cara mengelola data secara dinamis menggunakan linked list, dan implementasinya sangat relevan dalam konteks pemrograman berbasis struktur data.