LAPORAN PRAKTIKUM

Modul 4

“SINGLE LINKED LIST (BAGIAN PERTAMA)”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

- Memahami penggunaan linked list dengan pointer operator- operator dalam program.
- Memahami operasi-operasi dasar dalam linked list.
- Membuat program dengan menggunakan linked list dengan prototype yang ada

2. Landasan Teori

- Linked List dengan Pointer
  Linked List dengan Pointer adalah struktur data dinamis yang terdiri dari sekumpulan elemen yang saling terhubung. Setiap elemen dalam linked list disebut node, dan setiap node berisi dua bagian:
  - Data: Menyimpan informasi (misalnya nilai integer, string, atau objek lainnya).
  - Pointer: Menyimpan alamat memori dari node berikutnya di dalam list, sehingga membentuk hubungan antar-node.

  Pointer dalam linked list digunakan untuk menghubungkan node-node. Dengan pointer, setiap node dapat menunjuk ke node berikutnya di memori, meskipun node tersebut tidak disimpan secara berurutan dalam memori. Hal ini berbeda dari array, di mana elemen-elemen disimpan secara bersebelahan dalam memori. Pointer memberikan fleksibilitas karena memungkinkan linked list untuk tumbuh atau menyusut dengan mudah. Kita bisa menambahkan, menghapus, atau menyisipkan node tanpa perlu menggeser elemen-elemen lain, seperti yang sering terjadi pada array.

  Contoh Visualisasi:
  - Node dengan Data dan Pointer:
    Setiap node memiliki data dan pointer ke node berikutnya.
    [Data | Next] -> [Data | Next] -> [Data | Next] -> nullptr
    Jika Data = 5, Next = alamat memori node berikutnya.
  - Pointer head:
    Pointer head menunjuk ke node pertama.
    Jika tidak ada node dalam list, head = nullptr.
    Head -> [Data: 5 | Next] -> [Data: 10 | Next] -> nullptr

  Operasi Dasar dalam Linked List dengan Pointer:
  Insert Node: Menambahkan node ke linked list.
  Delete Node: Menghapus node dari linked list.
  Traverse: Melakukan traversal atau iterasi melalui node-node dalam linked list.

- Single Linked List
  Single Linked List Linked List adalah salah satu bentuk struktur data, berisi kumpulan data (node) yang tersusun secara sekuensial, saling sambungmenyambung, dinamis dan terbatas.
  - Linked List sering disebut juga Senarai Berantai
  - Linked List saling terhubung dengan bantuan variabel pointer

- Masing-masing data dalam Linked List disebut dengan node (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa struct yang terdiri dari beberapa field.
Single Linked List adalah sebuah LINKED LIST yang menggunakan sebuah variabel pointer saja untuk menyimpan banyak data dengan metode LINKED LIST, suatu daftar isi yang saling berhubungan.
Komponen dalam Single Linked List:
    - Node: Setiap elemen dalam linked list disebut node. Setiap node terdiri dari dua bagian yaitu, data untuk menyimpan nilai atau informasi yang ingin kita simpan (misalnya angka, string, objek, dll.). lalu, pointer (next) menyimpan alamat atau referensi ke node berikutnya dalam linked list. Node terakhir dalam linked list menunjuk ke nullptr, yang menandakan akhir dari list.
    - Pointer head: Merupakan pointer yang menunjuk ke node pertama dalam linked list. Jika list kosong, pointer head diatur ke nullptr.

    Visualisasi Single Linked List
    Head -> [Data: 10 | Next] -> [Data: 20 | Next] -> [Data: 30 | Next] -> nullptr
    Pada contoh di atas:
    Head menunjuk ke node pertama yang berisi data 10. Node pertama menunjuk ke node kedua yang berisi data 20. Kemudian node kedua menunjuk ke node ketiga yang berisi data 30. Sedangkan node ketiga adalah node terakhir dan menunjuk ke nullptr.
    Operasi Dasar pada Single Linked List:
    - Operasi insert pada Single Linked List memungkinkan kita untuk menyisipkan node baru pada berbagai posisi dalam list. Kita dapat menambahkan node di bagian depan (head), belakang (tail), atau di antara node-node yang sudah ada.
    - Operasi delete pada Single Linked List memungkinkan kita untuk menghapus node dari berbagai posisi dalam list. Kita dapat menghapus node pertama (head), node terakhir (tail), atau node yang mengandung nilai tertentu.
    - Operasi traversal pada Single Linked List melibatkan iterasi melalui setiap node mulai dari head (node pertama) hingga mencapai node dengan nilai next yang null (menandakan akhir list). Pada setiap node, kita dapat melakukan berbagai operasi, seperti mencetak data, membandingkan dengan nilai tertentu, atau melakukan perhitungan.

- Delete
  Delete merupakan salah satu operasi dasar dalam single linked list di mana node tertentu dihapus dari list.Ini bisa dilakukan dengan beberapa cara, tergantung pada posisi node yang ingin dihapus, seperti, menghapus node di depan (head), menghapus node di belakang (tail), dan Menghapus node berdasarkan nilai tertentu.

  Untuk menghapus node dari linked list, kita perlu melakukan traversal (penelusuran) hingga menemukan node yang ingin dihapus, kemudian mengatur

pointer untuk menghubungkan node sebelumnya ke node setelahnya, sehingga node yang dihapus tidak lagi berada dalam list.

Contoh Implementasi Operasi Delete dalam C++:
1. Menghapus Node di Depan (Delete from Front)
Penghapusan node di depan mengubah pointer head sehingga menunjuk ke node kedua. Node pertama kemudian dihapus.

```
void deleteFront(Node** head) {
  if (*head == nullptr) {
    cout << "Linked list kosong, tidak ada yang bisa dihapus." << endl;
    return;
  }

  Node* temp = *head; // Simpan node pertama dalam variabel sementara
  *head = (*head)->next; // Ubah head ke node kedua
  delete temp; // Hapus node pertama
}
```

2. Menghapus Node di Belakang (Delete from Back)
Penghapusan node di belakang memerlukan traversal hingga mencapai node terakhir dan node kedua dari belakang, kemudian pointer next node kedua dari belakang diatur ke nullptr.

```
void deleteBack(Node** head) {
  if (*head == nullptr) {
    cout << "Linked list kosong, tidak ada yang bisa dihapus." << endl;
    return;
  }

  if ((*head)->next == nullptr) { // Jika hanya ada satu node
    delete *head;
    *head = nullptr;
    return;
  }

  Node* temp = *head;
  while (temp->next->next != nullptr) { // Menelusuri hingga node kedua dari belakang
    temp = temp->next;
  }

  delete temp->next; // Hapus node terakhir
  temp->next = nullptr; // Node kedua dari belakang sekarang menjadi node terakhir
}
```

3. Menghapus Node dengan Nilai Tertentu (Delete by Value)
Untuk menghapus node dengan nilai tertentu, kita perlu menelusuri linked list sampai menemukan node yang berisi nilai tersebut, kemudian menghubungkan node sebelumnya ke node berikutnya.

```
void deleteByValue(Node** head, int key) {
  if (*head == nullptr) {
    cout << "Linked list kosong, tidak ada yang bisa dihapus." << endl;
    return;
  }

  Node* temp = *head;
  Node* prev = nullptr;

  // Jika node pertama berisi nilai yang ingin dihapus
  if (temp != nullptr && temp->data == key) {
    *head = temp->next; // Ubah head menjadi node berikutnya
    delete temp; // Hapus node pertama
    return;
  }

  // Menelusuri node hingga menemukan nilai
  while (temp != nullptr && temp->data != key) {
    prev = temp;
    temp = temp->next;
  }

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

  // Node ditemukan, hubungkan node sebelumnya dengan node setelahnya
  prev->next = temp->next;
  delete temp; // Hapus node
}
```

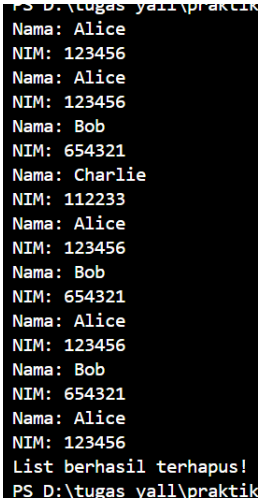
- Update
  Update pada Single Linked List (SLL) adalah operasi di mana kita mengganti nilai dari suatu node yang ada dalam linked list dengan nilai baru. Operasi ini melibatkan traversal (penelusuran) linked list untuk menemukan node yang ingin di-update, kemudian mengganti nilai data di dalam node tersebut.
  Langkah-langkah Update Node dalam Single Linked List:
  Traversal, yaitu perlu menelusuri linked list dari node pertama hingga menemukan node yang ingin di-update.
  Penggantian Nilai, setelah node yang ingin di-update ditemukan, nilai data dalam node tersebut diganti dengan nilai baru.

3. Guided

1) Program tersebut merupakan implementasi dari struktur data linked list. Program ini digunakan untuk menyimpan dan mengelola data mahasiswa, yang terdiri dari nama dan NIM. Struktur data di dalamnya ada struk mahasiswa dengan atribut ‘nama’ dan ‘nim’. Lalu, digunakan struk node yang mewakili elemen dari linked list berisi data mahasiswa dan pointer ke node berikutnya. Pada fungsi utama, Fungsi init() digunakan untuk memulai sebuah daftar kosong. Fungsi insertDepan() dan insertBelakang() memungkinkan kita menambahkan data baru ke awal atau akhir daftar. Fungsi hitungList() digunakan untuk menghitung jumlah data dalam daftar. Untuk menghapus data, kita dapat menggunakan hapusDepan() atau hapusBelakang(). Fungsi tampil() akan menampilkan seluruh data dalam daftar secara berurutan. Terakhir, fungsi clearList() berfungsi untuk mengosongkan seluruh daftar. Dengan fungsi-fungsi ini, kita dapat dengan mudah menambah, menghapus, dan melihat data yang tersimpan dalam linked list.

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

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

//struct node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// inisialisasi list
void init(){
    head = nullptr;
    tail = nullptr;
}

// pencegahan 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 (isimpty()){
        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;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus belakang
void hapusBelakang(){
    if (!isimpty()){
        if (head == tail){
            delete head;
            head = tail = nullptr;
        } 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 << "\nNIM: " << 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(m3);
    tampil();

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

    // menghapus seluruh list
    clearList();

    return 0;
}
```

Ouput dari program:



2) Program ini masih merupakan implementasi linked list, yang digunakan untuk menyimpan dan mengelola data dalam bentuk daftar yang dinamis. Dibuat struktur node untuk mewakili elemen dari linked list, yang berisi data bertipe integer dan pointer ke node berikutnya. Kemudian fungsi utama, terdapat mengalokasikan memori untuk node baru dan menginisialisasi data dengan nilai yang diberikan. Menghapus node dari memori. Setelah itu pengecekan apakah linked list kosong. Lalu ditambahkan elemen baru di awal linked list. Tambahkan elemen baru di akhir linked list dan tampilkan semua elemen dalam linked list. Hitung jumlah elemen dalam linked list dan hapus semua elemen dalam linked list dan dealokasi memori. Contoh penggunaan dengan menambahkan 3 elemen ke dalam linked list, tampilkan isi dan jumlah elemen serta hapus selurh list dan menampilkan hasil setelah di hapus.

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

// Definisi struktur untuk elemen list
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk mengalokasi memori untuk node baru
Node* alokasi(int value){
    Node* newNode = new Node;
    if (newNode != nullptr){
        newNode->data = value;
        newNode->next = nullptr;
    }
    return newNode;
}

// fungsi untuk dealokasi memori node
void dealokasi(Node* node){
    delete node;
}
// pengecekan apakah list kosong
bool isListEmpty(Node* head){
    return head == nullptr;
}

// menambahkan elemen diawal list
void insertFirst(Node* &head, int value){
    Node* newNode = alokasi(value);
    if (newNode != nullptr){
        newNode->next = head;
        head = newNode;
    }
}

// menambahkan elemen diakhir list
void insertLast(Node* &head, int value){
    Node* newNode = alokasi(value);
    if (newNode != nullptr){
        if (isListEmpty(head)) {
            head = newNode;
        }
    } else {
        Node* temp = head;
        while (temp->next != nullptr){
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
// menampilkan semua elemen dalam list
void printList(Node* head){
    if (isListEmpty(head)){
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr){
            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++;
        temp = temp->next;
    }
    return count;
}

// menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node* &head){
    while(head != nullptr){
        Node* temp = head;
        head = head->next;
        dealokasi(temp);
    }
}

int main(){
    Node* head = nullptr;

    //menambahkan elemen ke dalam list
    insertFirst(head, 10);
    insertLast(head, 20);
    insertLast(head, 30);

    //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 program:

```
PS D:\tugas yall\praktikum sd\pertemuan_4\d
Isi List: 10
Jumlah elemen: 1
Isi List setelah penghapusan: List kosong!
PS D:\tugas yall\praktikum sd\pertemuan_4\d
```

4. Unguided

1) Membuat Single Linked List dengan fungsi-fungsi dasar untuk insert node di depan, insert node di belakang, dan mencetak isi Linked List:
   - Struktur dasar node
     Dibuat sebuah node untuk menyimpan data pada node. Dalam hal ini, tipe datanya adalah int, tetapi bisa diubah menjadi tipe lain jika dibutuhkan. Pointer yang menunjuk ke node berikutnya, Jika node ini adalah node terakhir, maka next akan bernilai nullptr (penanda akhir list).
   - Fungsi menambahkan node di depan
     Fungsi ini membuat node baru untuk memasukkan data baru ke dalam node baru. Lalu, hubungkan node baru ke node sebelumnya agar terhubung ke node pertama yang saat ini ada di head. Head di update menjadi newNode, karena node baru ini sekarang adalah node pertama dalam list.
   - Fungsi menambakan node di belakang
     Seperti pada fungsi insertFront, node baru dibuat. Masukkan data baru ke dalam node baru. Atur pointer next: Karena node baru akan menjadi node terakhir. Dibuat kondisi ketika list kosong: Jika *head == nullptr, artinya linked list kosong, maka node baru langsung dijadikan head. Traversal ke node terakhir Jika linked list sudah ada isinya, kita perlu mencari node terakhir lalu node baru dihubungkan ke node terakhir tersebut.
   - Fungsi mencetak isi linked list
     Dibuat perulangan untuk melakukan iterasi melalui seluruh node dalam linked list menggunakan while. Setiap kali node dikunjungi, data pada node tersebut dicetak. Jika masih ada node berikutnya, program juga akan mencetak simbol ->. Setelah mencetak data, pointer node diperbarui untuk menunjuk ke node berikutnya. Ketika node mencapai nullptr, loop berhenti, dan program mencetak baris baru.
   - Fungsi utama program.
     Pointer head diinisialisasi dengan nullptr, yang menandakan bahwa linked list pada awalnya kosong. Ada tiga operasi insert dilakukan:
     insertFront(&head, 10): Menambahkan node dengan nilai 10 di depan.
     insertBack(&head, 20): Menambahkan node dengan nilai 20 di belakang.
     insertFront(&head, 5): Menambahkan node dengan nilai 5 di depan, sehingga list menjadi 5->10->20.
     Fungsi printList digunakan untuk mencetak isi dari linked list.

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

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

// Funngsi menambah node di depan
void insertFront(Node** head, int newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}

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

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

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

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

int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked list: ";
    printList(head);

    return 0;
}
```

Ouput program:

```
PS D:\tugas yall\praktikum
Linked list: 5-> 10-> 20
PS D:\tugas yall\praktikum
```

2) Menghapus node pada Linked List

Seperti biasanya dibuat terlebih dahulu struktur dasar node yang menyimpan data dan next. Kemudian, buat

- Fungsi menambahkan node di depan linked list
  Buat node baru yang akan mengalokasikan memori. Nilai newData dimasukkan ke dalam field data dari node baru. Node baru harus terhubung dengan node head yang ada saat ini, perbarui head karena node baru akan menjadi node pertama.
- Fungsi menambahkan node di belakang linked list
  Membuat node baru yang dialokasikan menggunakan new Node(). Isi nilai data dimasukkan ke dalam node baru. Karena node ini akan menjadi node terakhir, maka newNode->next di-set sebagai nullptr. Jika linked list kosong, node baru menjadi head. Jika linked list sudah ada isinya, program melakukan traversal melalui list sampai menemukan node terakhir. Setelah menemukan node terakhir, kita hubungkan temp->next ke newNode.
- Fungsi menghapus node berdasarkan nilai tertentu
  Untuk menghapus node dalam linked list, kita akan mencari node yang ingin dihapus, lalu memotong hubungannya dengan node-node lainnya. Kita juga perlu memperbarui pointer head jika node yang dihapus adalah node pertama. Proses ini melibatkan penelusuran seluruh linked list hingga node yang diinginkan ditemukan, kemudian menghapus node tersebut dari memori.
- Fungsi mencetak isi linked list
  Fungsi traversal bekerja seperti berjalan menyusuri sebuah daftar. Kita mulai dari awal daftar dan melihat nilai data pada setiap item dalam daftar itu. Setelah melihat satu item, kita akan pindah ke item berikutnya sampai kita mencapai akhir daftar. Untuk menunjukkan bahwa item-item tersebut saling terhubung, kita akan menambahkan tanda panah "->" di antara nilai data setiap item.
- Fungsi utama program
  Program ini memulai dengan membuat sebuah struktur data yang disebut linked list. Linked list ini awalnya kosong. Kemudian, kita menambahkan beberapa elemen data ke dalam linked list tersebut. Elemen-elemen data ini diurutkan dalam sebuah urutan tertentu. Setelah itu, kita menghapus salah satu elemen data dari linked list. Proses ini menunjukkan bagaimana kita dapat memanipulasi data dalam sebuah linked list, mulai dari membuat list kosong, menambahkan

elemen, hingga menghapus elemen.

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

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

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

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

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

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

// Fungsi untuk menghapus node dengan nilai tertentu
void deleteNode(Node** head, int key) {
    // Simpan node head sementara
    Node* temp = *head;
    Node* prev = nullptr;

    // Jika node yang akan dihapus adalah head
    if (temp != nullptr && temp->data == key) {
        *head = temp->next; // Pindahkan head ke node berikutnya
        delete temp; // Hapus node lama
        return;
    }

    // Cari node yang punya nilai tertentu
    while (temp != nullptr && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // kalau node tidak ditemukan
    if (temp == nullptr) return;

    // Lepaskan node dari linked list
    prev->next = temp->next;

    // Hapus node
    delete temp;
}

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

int main() {
    Node* head = nullptr;

    // Tambah beberapa node
    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

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

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

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

    return 0;
}
```

Output program:

```
PS D:\tugas yall\praktikum sd\Unguided mod 4
Linked list sebelum penghapusan: 5-> 10-> 20
Linked list setelah penghapusan: 5-> 20
PS D:\tugas yall\praktikum sd\Unguided mod 4
```

3) Mencari dan menghitung Panjang Linked List
   Membuat struktur dasar node terlebih dahulu lalu,
   - Fungsi mencari node dengan nilai tertentu
     pencarian pada linked list menggunakan pendekatan linear search. Pointer current bertindak sebagai iterator yang bergerak secara berurutan dari node pertama ke node terakhir. Pada setiap iterasi, nilai data pada node current dibandingkan dengan nilai yang dicari. Jika terdapat kecocokan, pencarian dihentikan dan fungsi mengembalikan nilai benar. Jika tidak ada kecocokan pada akhir iterasi, fungsi mengembalikan nilai salah.
   - Fungsi menghitung Panjang linked list
     penghitungan jumlah node pada linked list menggunakan pendekatan iteratif. Variabel count bertindak sebagai akumulator yang menyimpan jumlah node yang telah dikunjungi. Proses iterasi dimulai dari node head dan berlanjut hingga node terakhir (null). Pada setiap iterasi, nilai count ditambah satu dan pointer bergerak ke node berikutnya. Setelah proses iterasi selesai, nilai count yang diperoleh merupakan jumlah total node dalam linked list.
   - Fungsi utama
     Proses inisialisasi dilakukan dengan mengatur pointer head ke null. Penambahan node dilakukan dengan mengalokasikan memori untuk node baru dan menghubungkannya dengan node sebelumnya. Pencarian node dilakukan dengan menelusuri seluruh list secara berurutan hingga node yang dicari ditemukan atau akhir list tercapai. Penghitungan jumlah node dilakukan dengan menelusuri seluruh list dan menghitung setiap node yang dijumpai.

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

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

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

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

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

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

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

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

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

int main() {
    Node* head = nullptr;

    // Tambah beberapa node
    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

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

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

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

    return 0;
}
```

Output program:

```
PS D:\tugas yall\praktikum sd\U
Isi linked list: 5-> 10-> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS D:\tugas yall\praktikum sd\U
```

5. Kesimpulan

Single Linked List adalah cara untuk menyimpan data yang saling terhubung satu sama lain. Setiap node memiliki dua bagian, yaitu data yang menyimpan informasi dan pointer yang menunjuk ke node berikutnya. Berbeda dengan array, linked list memungkinkan pengelolaan data secara lebih fleksibel karena penambahan dan penghapusan elemen tidak memerlukan pergeseran elemen lainnya. Operasi dasar dalam linked list meliputi penambahan (insert), penghapusan (delete), dan traversal (penelusuran) untuk mengiterasi node-node dalam list. Dalam implementasinya, linked list berguna dalam pengelolaan data yang dinamis seperti menyimpan dan mengelola data mahasiswa. Program ini memungkinkan penambahan, penghapusan, dan penelusuran data dengan mudah melalui fungsi-fungsi dasar seperti insertDepan, insertBelakang, dan deleteNode.