

LAPORAN PRAKTIKUM

Modul 4

Single Linked List



Disusun Oleh:

Ryan Gabriel Togar Simamora (2311104045)

Kelas : SE0702

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. 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

II. Landasan Teori

1. Linked List dengan Pointer

Linked list adalah salah satu struktur data linier yang terdiri dari serangkaian node, di mana setiap node menyimpan data dan pointer yang menunjuk ke node berikutnya. Dalam implementasi linked list, kita dapat menggunakan array atau pointer.

Namun, penggunaan pointer lebih sering dipilih karena beberapa alasan berikut:

a. Array bersifat statis, sedangkan pointer dinamis.

Array memiliki ukuran tetap yang harus ditentukan di awal program, sedangkan pointer memungkinkan penggunaan memori secara dinamis. Ini membuat linked list lebih fleksibel, karena ukurannya dapat bertambah atau berkurang sesuai kebutuhan saat program berjalan.

b. Bentuk data dalam linked list saling berhubungan.

Setiap node dalam linked list terhubung satu sama lain melalui pointer, yang membuat penggunaan pointer lebih alami dibandingkan array. Pointer langsung mengarahkan ke node berikutnya, sementara dalam array, indeks harus dihitung secara manual.

c. Fleksibilitas linked list cocok dengan sifat pointer.

Pointer memungkinkan kita untuk dengan mudah mengatur ulang node dalam linked list tanpa perlu melakukan operasi kompleks seperti penggeseran elemen, yang umum dalam array.

d. Array lebih sulit dalam menangani linked list.

Ketika menggunakan array untuk mengelola linked list, operasi seperti penyisipan dan penghapusan elemen akan memerlukan penggeseran elemen lain, yang tidak efisien. Sedangkan, dengan pointer, kita hanya perlu memperbarui beberapa referensi untuk menyisipkan atau menghapus elemen.

e. Array lebih cocok untuk kumpulan data dengan ukuran tetap.

Jika jumlah elemen sudah diketahui sejak awal dan tidak akan berubah, array lebih efisien. Namun, jika jumlah elemen berubah-ubah atau tidak diketahui, linked list dengan pointer adalah pilihan yang lebih baik.

2. Sifat dari Single Linked List

Single Linked List memiliki karakteristik yang membedakannya dari struktur data lainnya, yaitu:

a. Hanya memerlukan satu pointer.

Setiap node dalam Single Linked List hanya menyimpan satu pointer, yaitu pointer ke node berikutnya (bukan dua, seperti dalam double linked list).

b. Node akhir menunjuk ke Nil.

Node terakhir dalam Single Linked List akan menunjuk ke nilai Nil atau nullptr, yang menandakan akhir dari list. Dalam list circular, node terakhir menunjuk kembali ke node pertama.

c. Pembacaan hanya dapat dilakukan secara maju.

Single Linked List hanya memungkinkan traversal (perjalanan) maju dari node pertama hingga node terakhir. Untuk traversal mundur, diperlukan struktur data lain seperti double linked list.

d. Pencarian dilakukan secara sequensial.

Jika data dalam list tidak terurut, pencarian elemen tertentu harus dilakukan dengan memeriksa setiap node satu per satu dari awal hingga akhir.

e. Penyisipan dan penghapusan lebih mudah.

Penyisipan atau penghapusan elemen di tengah linked list lebih mudah dilakukan dibandingkan array, karena hanya perlu mengatur ulang pointer tanpa menggeser elemen lain.

3. Komponen-Komponen dalam Linked List

Pembentukan Single Linked List melibatkan beberapa komponen utama yang perlu diperhatikan:

a. Pembentukan List:

Membuat linked list dimulai dengan mendeklarasikan sebuah pointer (biasanya disebut head) yang menunjuk ke node pertama dalam list.

b. Pengalokasian Memori:

Di dalam C++, memori untuk node baru dialokasikan menggunakan fungsi new. Setiap kali elemen baru ditambahkan ke list, memori baru harus dialokasikan untuk node tersebut.

Contoh:

```
Node* newNode = new Node();
```

Dealokasi Memori:

Setelah node tidak lagi diperlukan, memori yang dialokasikan harus dikembalikan ke sistem menggunakan delete untuk mencegah kebocoran memori.

Contoh:

```
delete nodeToDelete;
```

Pengecekan List:

Sebelum melakukan operasi pada list, sering kali diperlukan pengecekan apakah list kosong (misalnya, head menunjuk ke nullptr) atau sudah berisi node.

4. Operasi-Operasi pada Single Linked List

a. Insert (Penyisipan Elemen):

Penyisipan elemen dalam linked list dapat dilakukan di awal, akhir, atau setelah node tertentu :

- ❖ **Insert First:** Menyisipkan elemen di awal list dengan membuat node baru dan mengarahkan pointer next dari node baru ke node pertama yang lama.
- ❖ **Insert Last:** Menyisipkan elemen di akhir list memerlukan traversing dari awal hingga node terakhir, lalu menambahkan node baru di akhir.
- ❖ **Insert After:** Menyisipkan elemen setelah node tertentu dengan memperbarui pointer dari node tersebut dan node baru.

b. View (Menampilkan Elemen):

Untuk menampilkan seluruh elemen dalam linked list, traversal dilakukan dari node pertama hingga node terakhir, mencetak data di setiap node.

c. Delete (Penghapusan Elemen):

Penghapusan elemen juga dapat dilakukan di awal, akhir, atau setelah node tertentu:

- ❖ **Delete First:** Menghapus node pertama dan mengalihkan pointer head ke node kedua.
- ❖ **Delete Last:** Menghapus node terakhir memerlukan traversing hingga node sebelum terakhir, lalu menghapus node terakhir.
- ❖ **Delete After:** Menghapus node setelah node tertentu dengan memperbarui pointer next dari node tersebut.

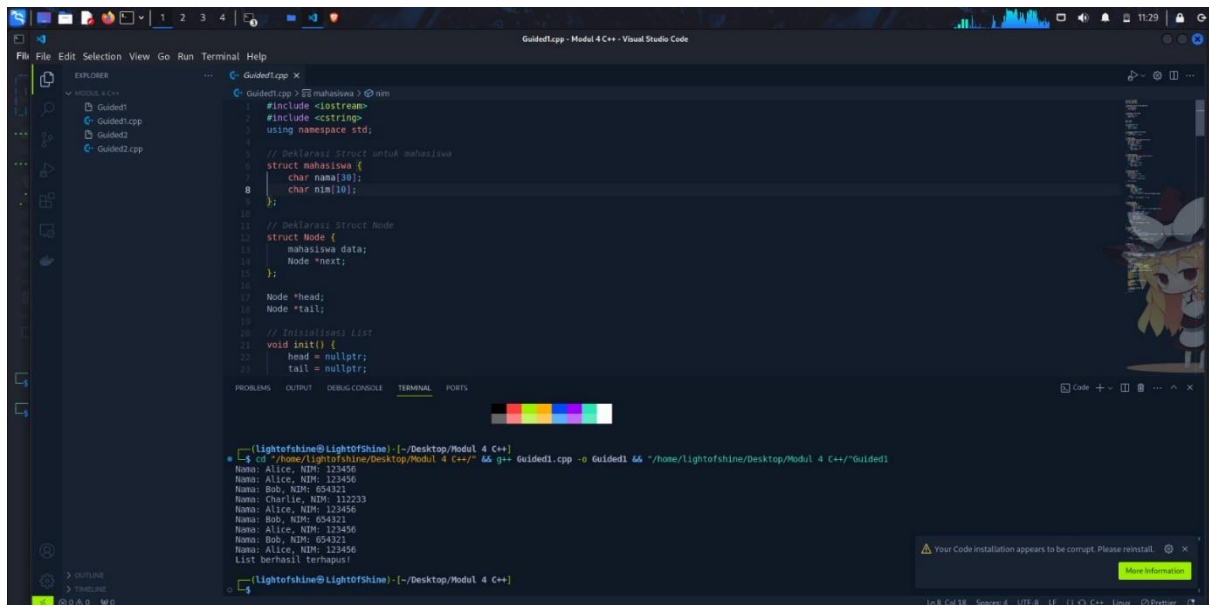
d. Update (Memperbarui Elemen):

Operasi update memungkinkan untuk mengubah nilai dari data dalam node tertentu.

Single Linked List adalah struktur data dinamis yang sangat fleksibel dan efisien untuk menangani data yang memerlukan perubahan ukuran pada runtime. Dengan menggunakan pointer, linked list memungkinkan pengelolaan memori yang lebih baik dibandingkan array, terutama untuk penyisipan dan penghapusan elemen. Operasi dasar seperti insert, delete, view, dan update dapat diimplementasikan dengan mudah dalam bahasa C++ menggunakan pointer dan alokasi dinamis.

III. Guided

File Guided1.cpp



```
#include <iostream>
#include <string>
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;
}

(Lightofshin@LightofShin) ~/Desktop/Modul 4 C++
$ cd "/home/lightofshin/Desktop/Modul 4 C++/" && g++ Guided1.cpp -o Guided1 && ./Guided1
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Charlie, NIM: 123233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
Nama: Alice, NIM: 123456
List berhasil terhapus!
```

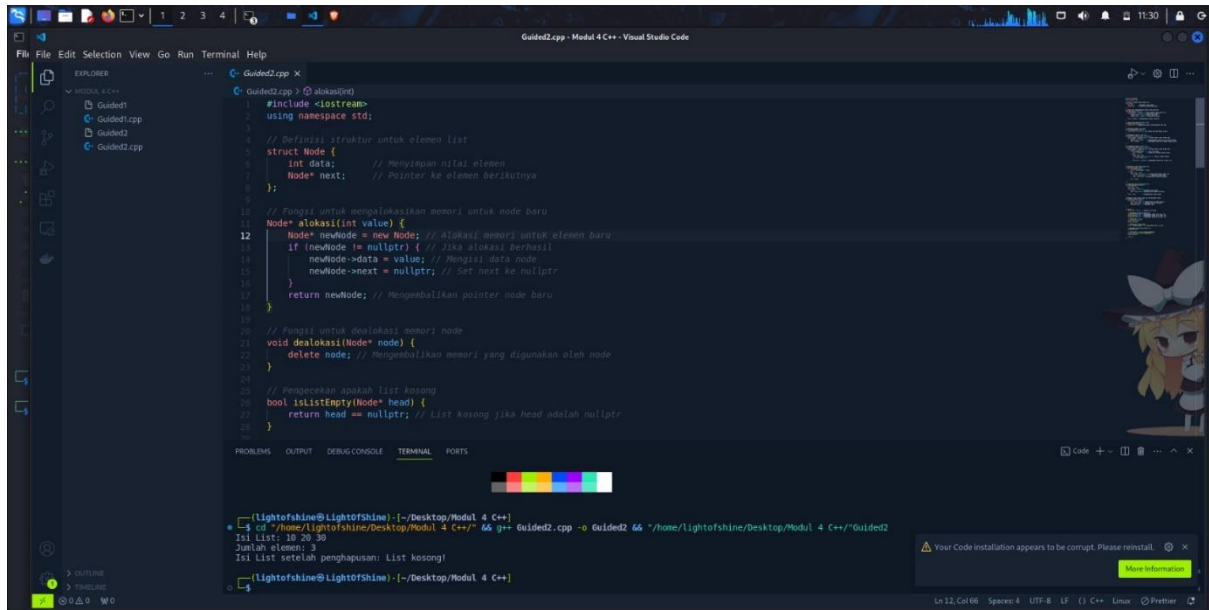
Untuk Source Codenya Lebih Lengkap Dibawah ini :

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 // Deklarasi Struct untuk mahasiswa
6 struct mahasiswa {
7     char nama[50];
8     char nim[50];
9 };
10
11 // Deklarasi Struct Node
12 struct Node {
13     mahasiswa data;
14     Node *next;
15 };
16
17 Node *head;
18 Node *tail;
19
20 // Inisialisasi List
21 void Init() {
22     head = nullptr;
23     tail = nullptr;
24 }
25
26 // Pengecekan apakah list kosong
27 bool isEmpty() {
28     return head == nullptr;
29 }
30
31 // Tambah Depan
32 void InsertDepan(const mahasiswa &data) {
33     Node *baru = new Node;
34     baru->data = data;
35     baru->next = nullptr;
36     if (isEmpty()) {
37         head = tail = baru;
38     } else {
39         baru->next = head;
40         head = baru;
41     }
42 }
43
44 // Tambah Belakang
45 void InsertBelakang(const mahasiswa &data) {
46     Node *baru = new Node;
47     baru->data = data;
48     baru->next = nullptr;
49     if (isEmpty()) {
50         head = tail = baru;
51     } else {
52         tail->next = baru;
53         tail = baru;
54     }
55 }
56
57 // Hitung Jumlah List
58 int hitungList() {
59     Node *current = head;
60     int jumlah = 0;
61     while (current != nullptr) {
62         jumlah++;
63         current = current->next;
64     }
65     return jumlah;
66 }
67
68 // Hapus Depan
69 void hapusDepan() {
70     if (!isEmpty()) {
71         Node *hapus = head;
72         head = head->next;
73         delete hapus;
74         if (head == nullptr) {
75             tail = nullptr; // Jika list menjadi kosong
76         }
77     } else {
78         cout << "List kosong!" << endl;
79     }
80 }
81
82 // Hapus Belakang
83 void hapusBelakang() {
84     if (!isEmpty()) {
85         if (head == tail) {
86             delete head;
87             head = tail = nullptr; // List menjadi kosong
88         } else {
89             Node *bantu = head;
90             while (bantu->next != tail) {
91                 bantu = bantu->next;
92             }
93             delete tail;
94             tail = bantu;
95             tail->next = nullptr;
96         }
97     } else {
98         cout << "List kosong!" << endl;
99     }
100 }
101
102 // Tampilkan list
103 void tampil() {
104     Node *current = head;
105     if (!isEmpty()) {
106         while (current != nullptr) {
107             cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
108             current = current->next;
109         }
110     } else {
111         cout << "List masih kosong!" << endl;
112     }
113 }
114
115 // Hapus List
116 void clearList() {
117     Node *current = head;
118     while (current != nullptr) {
119         Node *hapus = current;
120         current = current->next;
121         delete hapus;
122     }
123     head = tail = nullptr;
124     cout << "List berhasil terhapus!" << endl;
125 }
126
127 // Main function
128 int main() {
129     Init();
130
131     // Contoh data mahasiswa
132     mahasiswa m1 = {"Alice", "123456"};
133     mahasiswa m2 = {"Bob", "654321"};
134     mahasiswa m3 = {"Charlie", "112233"};
135
136     // Menambahkan mahasiswa ke dalam list
137     insertDepan(m1);
138     tampil();
139     insertBelakang(m2);
140     tampil();
141     insertDepan(m3);
142     tampil();
143
144     // Menghapus elemen dari list
145     hapusDepan();
146     tampil();
147     hapusBelakang();
148     tampil();
149
150     // Menghapus seluruh list
151     clearList();
152
153     return 0;
154 }

```

File Guided2.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
}

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

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

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

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

int main() {
    Node* head = nullptr; // Inisialisasi 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;
}
```

Lightofshine@Lightofshine:~/Desktop/Modul 4 C++
\$ cd ~/home/Lightofshine/Desktop/Modul 4 C++/ 66 g++ Guided2.cpp -o Guided2 66 ./Guided2
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
Lightofshine@Lightofshine:~/Desktop/Modul 4 C++

Untuk Source codenya lebih lengkapnya dibawah ini :



```
#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
}

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

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

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

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

int main() {
    Node* head = nullptr; // Inisialisasi 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;
}
```

IV. 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:

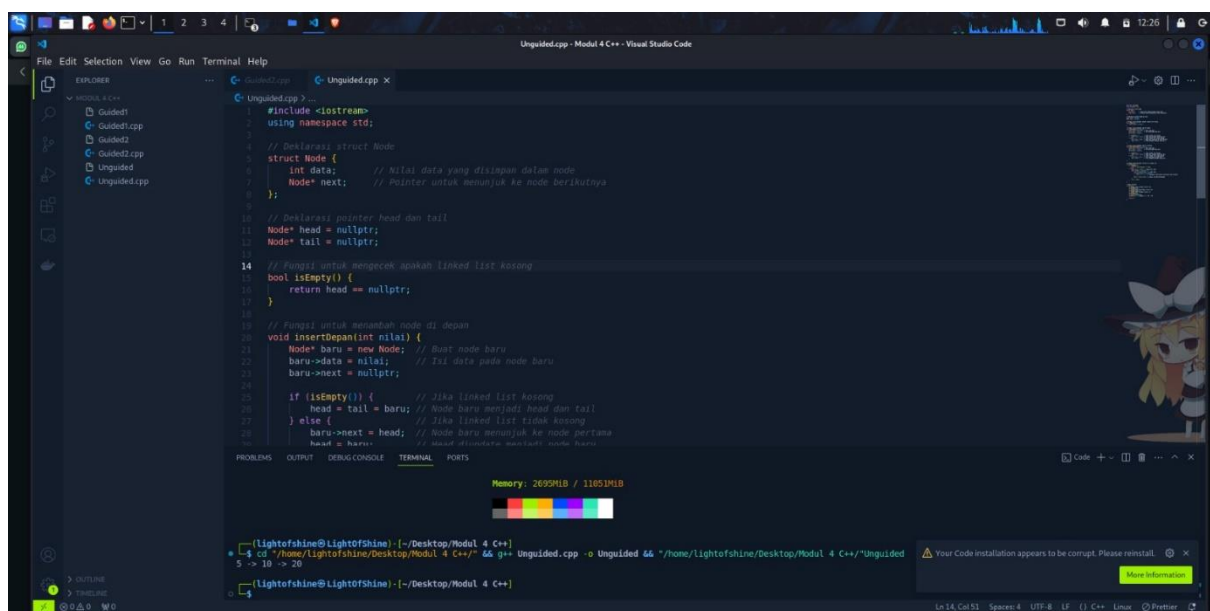
Input:

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

Jawab :



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

// Deklarasi struct Node
struct Node {
    int data; // Nilai data yang disimpan dalam node
    Node* next; // Pointer untuk menunjuk ke node berikutnya
};

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

// Fungsi untuk mengecek apakah linked list kosong
bool isEmpty() {
    return head == nullptr;
}

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

    if (isEmpty()) { // Jika linked list kosong
        head = tail = baru; // Node baru menjadi head dan tail
    } else { // Jika linked list tidak kosong
        baru->next = head; // Node baru menunjuk ke node pertama
        head = baru; // Head sekarang menjadi node baru
    }
}

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

    if (isEmpty()) { // Jika linked list kosong
        head = tail = baru; // Node baru menjadi head dan tail
    } else { // Jika linked list tidak kosong
        tail->next = baru; // Node baru menunjuk ke node terakhir
        tail = baru; // Tail sekarang menjadi node baru
    }
}

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

int main() {
    insertDepan(5);
    insertBelakang(10);
    insertDepan(20);
    cetakList();
    return 0;
}
```

Memory: 2695MB / 1105MB

```
Lightofshime@Lightofshime: ~/Desktop/Modul 4 C++
$ cd /home/Lightofshime/Desktop/Modul 4 C++/ && g++ Unguided.cpp -o Unguided && ./Unguided
5 -> 10 -> 20
Lightofshime@Lightofshime: ~/Desktop/Modul 4 C++
```


Untuk Source codenya lebih lengkapnya dibawah ini :

```
1  #include <iostream>
2  using namespace std;
3
4  // Deklarasi struct Node
5  struct Node {
6      int data;        // Nilai data yang disimpan dalam node
7      Node* next;      // Pointer untuk menunjuk ke node berikutnya
8  };
9
10 // Deklarasi pointer head dan tail
11 Node* head = nullptr;
12 Node* tail = nullptr;
13
14 // Fungsi untuk mengecek apakah linked list kosong
15 bool isEmpty() {
16     return head == nullptr;
17 }
18
19 // Fungsi untuk menambah node di depan
20 void insertDepan(int nilai) {
21     Node* baru = new Node; // Buat node baru
22     baru->data = nilai;    // Isi data pada node baru
23     baru->next = nullptr;
24
25     if (isEmpty()) { // Jika linked list kosong
26         head = tail = baru; // Node baru menjadi head dan tail
27     } else { // Jika linked list tidak kosong
28         baru->next = head; // Node baru menunjuk ke node pertama
29         head = baru;      // Head diupdate menjadi node baru
30     }
31 }
32
33 // Fungsi untuk menambah node di belakang
34 void insertBelakang(int nilai) {
35     Node* baru = new Node; // Buat node baru
36     baru->data = nilai;    // Isi data pada node baru
37     baru->next = nullptr;
38
39     if (isEmpty()) { // Jika linked list kosong
40         head = tail = baru; // Node baru menjadi head dan tail
41     } else { // Jika linked list tidak kosong
42         tail->next = baru; // Tail menunjuk ke node baru
43         tail = baru;      // Tail diupdate menjadi node baru
44     }
45 }
46
47 // Fungsi untuk mencetak seluruh isi linked list
48 void cetakList() {
49     if (isEmpty()) {
50         cout << "List kosong!" << endl;
51     } else {
52         Node* bantu = head; // Mulai dari head
53         while (bantu != nullptr) {
54             cout << bantu->data; // Cetak data
55             if (bantu->next != nullptr) {
56                 cout << " -> "; // Tambahkan tanda panah jika bukan node terakhir
57             }
58             bantu = bantu->next; // Lanjut ke node berikutnya
59         }
60         cout << endl;
61     }
62 }
63
64 // Main function
65 int main() {
66     // Tambah node di depan (nilai: 10)
67     insertDepan(10);
68     // Tambah node di belakang (nilai: 20)
69     insertBelakang(20);
70     // Tambah node di depan (nilai: 5)
71     insertDepan(5);
72     // Cetak linked list
73     cetakList(); // Output: 5 -> 10 -> 20
74
75     return 0;
76 }
```

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:

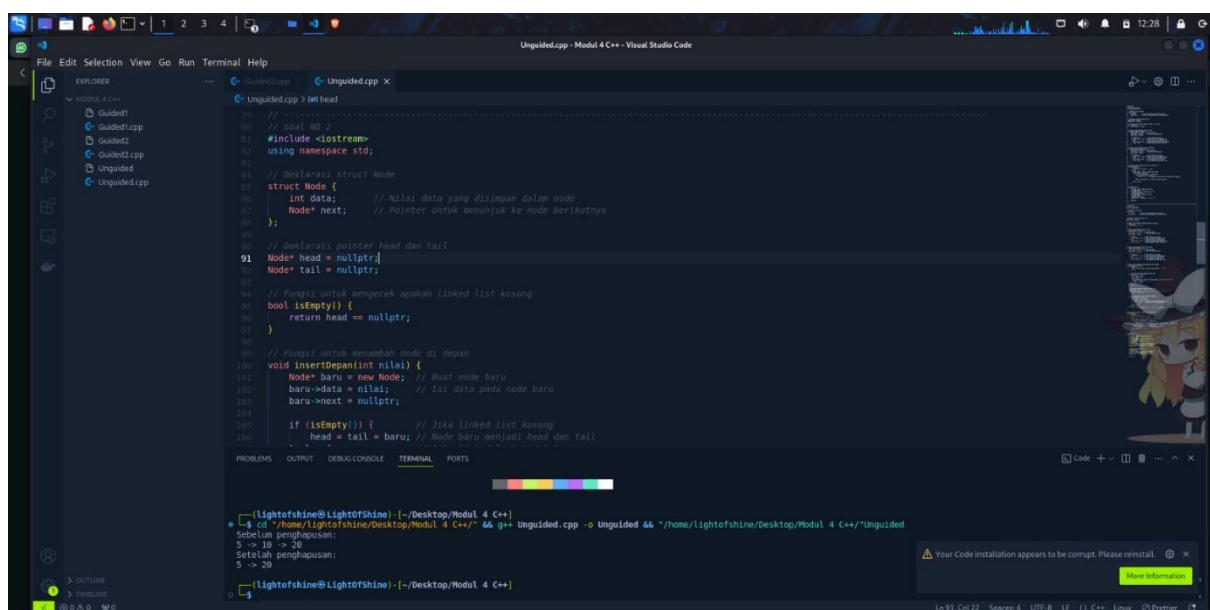
Input:

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

Jawab :



```
Unguided.cpp - Modul 4 C++ - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  Modul 4 C++
    Guided1
    Guided1.cpp
    Guided2
    Guided2.cpp
    Unguided
    Unguided.cpp
  Guided1.cpp
  Guided2.cpp
  Unguided.cpp
  Unguided.cpp X
  Unguided.cpp > Del head
35 //-----
36 // Soal NO 2
37 #include <iostream>
38 using namespace std;
39
40 // Deklarasi struct Node
41 struct Node {
42     // Nilai data yang disimpan dalam node
43     int data;
44     Node* next; // Pointer untuk menunjuk ke node berikutnya
45 };
46
47 // Deklarasi pointer head dan tail
48 Node* head = nullptr;
49 Node* tail = nullptr;
50
51 // Fungsi untuk mengecek apakah linked list kosong
52 bool isEmpty() {
53     return head == nullptr;
54 }
55
56 // Fungsi untuk menambah node di depan
57 void insertDepan(int nilai) {
58     Node* baru = new Node; // Buat node baru
59     baru->data = nilai; // Isi data pada node baru
60     baru->next = head;
61     if (isEmpty()) {
62         head = tail = baru; // Jika linked list kosong
63     } else {
64         head = baru; // Node baru menjadi head dan tail
65     }
66 }
67
68 // Fungsi untuk menghapus node dengan nilai tertentu
69 void deleteNode(int nilai) {
70     if (isEmpty()) {
71         cout << "Linked list kosong." << endl;
72         return;
73     }
74     if (head->data == nilai) {
75         head = head->next;
76     } else {
77         Node* temp = head;
78         while (temp->next != nullptr) {
79             if (temp->next->data == nilai) {
80                 temp->next = temp->next->next;
81                 delete temp->next;
82                 return;
83             }
84             temp = temp->next;
85         }
86     }
87     cout << "Node dengan nilai " << nilai << " telah dihapus." << endl;
88 }
89
90 // Fungsi untuk mencetak linked list
91 void printList() {
92     if (isEmpty()) {
93         cout << "Linked list kosong." << endl;
94         return;
95     }
96     Node* temp = head;
97     while (temp != nullptr) {
98         cout << temp->data << " ";
99         temp = temp->next;
100     }
101     cout << endl;
102 }
103
104 int main() {
105     int pilihan;
106     while (pilihan != 0) {
107         cout << "Pilih operasi yang akan dilakukan: " << endl;
108         cout << "1. Tambah node di depan (nilai: 10)" << endl;
109         cout << "2. Tambah node di belakang (nilai: 20)" << endl;
110         cout << "3. Tambah node di depan (nilai: 5)" << endl;
111         cout << "4. Hapus node dengan nilai (nilai: 10)" << endl;
112         cout << "5. Cetak linked list" << endl;
113         cout << "0. Exit" << endl;
114         int input;
115         while (input < 0 || input > 5) {
116             input = 0;
117             cout << "Masukkan pilihan yang valid (0-5): " << endl;
118         }
119         switch (input) {
120             case 1: insertDepan(10); break;
121             case 2: insertDepan(20); break;
122             case 3: insertDepan(5); break;
123             case 4: deleteNode(10); break;
124             case 5: printList(); break;
125             case 0: return 0;
126         }
127     }
128 }
```

```
(Lightofshine@Lightofshine) ~/Desktop/Modul 4 C++
$ cd ~/home/Lightofshine/Desktop/Modul 4 C++/ && g++ Unguided.cpp -o Unguided && ./Unguided
Setelah penambahan:
5 -> 10 -> 20
Setelah penghapusan:
5 -> 20
(Lightofshine@Lightofshine) ~/Desktop/Modul 4 C++
```

Untuk Source codenya lebih lengkapnya dibawah ini :

```
1 // Soal NO 2
2 #include <iostream>
3 using namespace std;
4
5 // Deklarasi struct Node
6 struct Node {
7     int data;           // Nilai data yang disimpan dalam node
8     Node* next;         // Pointer untuk menunjuk ke node berikutnya
9 };
10
11 // Deklarasi pointer head dan tail
12 Node* head = nullptr;
13 Node* tail = nullptr;
14
15 // Fungsi untuk mengecek apakah Linked list kosong
16 bool isEmpty() {
17     return head == nullptr;
18 }
19
20 // Fungsi untuk menambah node di depan
21 void insertDepan(int nilai) {
22     Node* baru = new Node; // Buat node baru
23     baru->data = nilai;     // Isi data pada node baru
24     baru->next = nullptr;
25
26     if (isEmpty()) {
27         head = tail = baru; // Node baru menjadi head dan tail
28     } else {
29         baru->next = head; // Node baru menunjuk ke node pertama
30         head = baru;      // Head diupdate menjadi node baru
31     }
32 }
33
34 // Fungsi untuk menambah node di belakang
35 void insertBelakang(int nilai) {
36     Node* baru = new Node; // Buat node baru
37     baru->data = nilai;     // Isi data pada node baru
38     baru->next = nullptr;
39
40     if (isEmpty()) {
41         head = tail = baru; // Node baru menjadi head dan tail
42     } else {
43         tail->next = baru; // Tail menunjuk ke node baru
44         tail = baru;      // Tail diupdate menjadi node baru
45     }
46 }
47
48 // Fungsi untuk menghapus node dengan nilai tertentu
49 void hapusNode(int nilai) {
50     if (isEmpty()) {
51         cout << "List kosong, tidak ada yang dihapus!" << endl;
52         return;
53     }
54
55     // Jika node yang akan dihapus adalah head
56     if (head->data == nilai) {
57         Node* hapus = head;
58         head = head->next;
59         delete hapus;
60         // Jika hanya ada 1 node dan sudah dihapus, tail juga harus di-null
61         if (head == nullptr) {
62             tail = nullptr;
63         }
64         return;
65     }
66
67     // Cari node yang akan dihapus (selain head)
68     Node* bantu = head;
69     while (bantu->next != nullptr && bantu->next->data != nilai) {
70         bantu = bantu->next;
71     }
72
73     // Jika node dengan nilai tersebut ditemukan
74     if (bantu->next != nullptr) {
75         Node* hapus = bantu->next;
76         bantu->next = hapus->next;
77         if (hapus == tail) { // Jika node yang dihapus adalah tail
78             tail = bantu;   // Update tail ke node sebelumnya
79         }
80         delete hapus;
81     } else {
82         cout << "Node dengan nilai " << nilai << " tidak ditemukan!" << endl;
83     }
84 }
85
86 // Fungsi untuk mencetak seluruh isi linked list
87 void cetakList() {
88     if (isEmpty()) {
89         cout << "List kosong!" << endl;
90     } else {
91         Node* bantu = head; // Mulai dari head
92         while (bantu != nullptr) {
93             cout << bantu->data; // Cetak data
94             if (bantu->next != nullptr) {
95                 cout << " -> "; // Tambahkan tanda panah jika bukan node terakhir
96             }
97             bantu = bantu->next; // Lanjut ke node berikutnya
98         }
99         cout << endl;
100     }
101 }
102
103 // Main function
104 int main() {
105     // Tambah node di depan (nilai: 10)
106     insertDepan(10);
107     // Tambah node di belakang (nilai: 20)
108     insertBelakang(20);
109     // Tambah node di depan (nilai: 5)
110     insertDepan(5);
111     // Cetak linked list sebelum penghapusan
112     cout << "Sebelum penghapusan:" << endl;
113     cetakList(); // Output: 5 -> 10 -> 20
114
115     // Hapus node dengan nilai 10
116     hapusNode(10);
117     // Cetak linked list setelah penghapusan
118     cout << "Setelah penghapusan:" << endl;
119     cetakList(); // Output: 5 -> 20
120
121     return 0;
122 }
```

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:

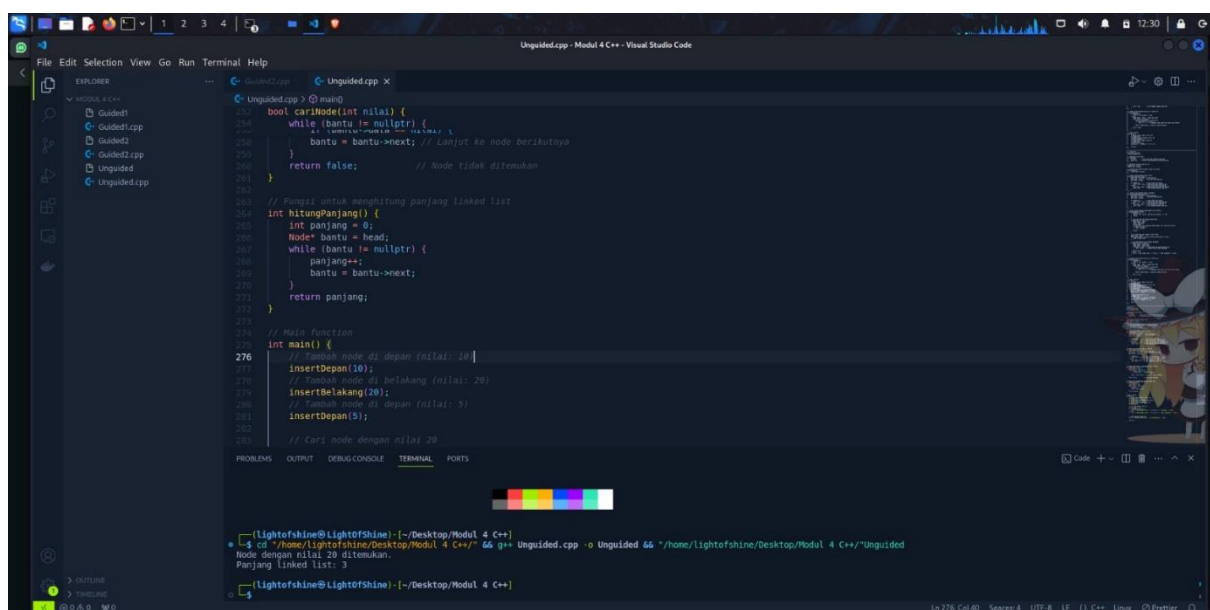
1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cari node dengan nilai 20
5. Cetak panjang linked list

Output:

Node dengan nilai 20 ditemukan.

Panjang linked list: 3

Jawab :



```
213 bool cariNode(int nilai) {
214     while (bantu != nullptr) {
215         if (bantu->data == nilai) return true;
216         bantu = bantu->next; // Lanjut ke node berikutnya
217     }
218     return false; // Node tidak ditemukan
219 }
220
221 // Fungsi untuk menghitung panjang linked list
222 int hitungPanjang() {
223     int panjang = 0;
224     Node* bantu = head;
225     while (bantu != nullptr) {
226         panjang++;
227         bantu = bantu->next;
228     }
229     return panjang;
230 }
231
232 // Main Function
233 int main() {
234     // Tambah node di depan (nilai: 10)
235     insertDepan(10);
236     // Tambah node di belakang (nilai: 20)
237     insertBelakang(20);
238     // Tambah node di depan (nilai: 5)
239     insertDepan(5);
240
241     // Cari node dengan nilai 20
242     if (cariNode(20)) {
243         cout << "Node dengan nilai 20 ditemukan." << endl;
244     } else {
245         cout << "Node dengan nilai 20 tidak ditemukan." << endl;
246     }
247
248     // Hitung panjang linked list
249     int panjang = hitungPanjang();
250     cout << "Panjang linked list: " << panjang << endl;
251 }
```

Output in terminal:

```
Lightfshine@Lightfshine:~/Desktop/Modul 4 C++$ g++ Unguided.cpp -o Unguided && ./Unguided
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```

Untuk Source codenya lebih lengkapnya dibawah ini :

```
1 // Soal NO 3
2 #include <iostream>
3 using namespace std;
4
5 // Deklarasi struct Node
6 struct Node {
7     int data;        // Nilai data yang disimpan dalam node
8     Node* next;      // Pointer untuk menunjuk ke node berikutnya
9 };
10
11 // Deklarasi pointer head dan tail
12 Node* head = nullptr;
13 Node* tail = nullptr;
14
15 // Fungsi untuk mengecek apakah linked list kosong
16 bool isEmpty() {
17     return head == nullptr;
18 }
19
20 // Fungsi untuk menambah node di depan
21 void insertDepan(int nilai) {
22     Node* baru = new Node; // Buat node baru
23     baru->data = nilai;    // Isi data pada node baru
24     baru->next = nullptr;
25
26     if (isEmpty()) { // Jika linked list kosong
27         head = tail = baru; // Node baru menjadi head dan tail
28     } else { // Jika linked list tidak kosong
29         baru->next = head; // Node baru menunjuk ke node pertama
30         head = baru; // Head diupdate menjadi node baru
31     }
32 }
33
34 // Fungsi untuk menambah node di belakang
35 void insertBelakang(int nilai) {
36     Node* baru = new Node; // Buat node baru
37     baru->data = nilai;    // Isi data pada node baru
38     baru->next = nullptr;
39
40     if (isEmpty()) { // Jika linked list kosong
41         head = tail = baru; // Node baru menjadi head dan tail
42     } else { // Jika linked list tidak kosong
43         tail->next = baru; // Tail menunjuk ke node baru
44         tail = baru; // Tail diupdate menjadi node baru
45     }
46 }
47
48 // Fungsi untuk mencari node dengan nilai tertentu
49 bool cariNode(int nilai) {
50     Node* bantu = head;
51     while (bantu != nullptr) {
52         if (bantu->data == nilai) {
53             return true; // Node ditemukan
54         }
55         bantu = bantu->next; // Lanjut ke node berikutnya
56     }
57     return false; // Node tidak ditemukan
58 }
59
60 // Fungsi untuk menghitung panjang linked list
61 int hitungPanjang() {
62     int panjang = 0;
63     Node* bantu = head;
64     while (bantu != nullptr) {
65         panjang++;
66         bantu = bantu->next;
67     }
68     return panjang;
69 }
70
71 // Main function
72 int main() {
73     // Tambah node di depan (nilai: 10)
74     insertDepan(10);
75     // Tambah node di belakang (nilai: 20)
76     insertBelakang(20);
77     // Tambah node di depan (nilai: 5)
78     insertDepan(5);
79
80     // Cari node dengan nilai 20
81     int nilaiCari = 20;
82     if (cariNode(nilaiCari)) {
83         cout << "Node dengan nilai " << nilaiCari << " ditemukan." << endl;
84     } else {
85         cout << "Node dengan nilai " << nilaiCari << " tidak ditemukan." << endl;
86     }
87
88     // Cetak panjang linked list
89     cout << "Panjang linked list: " << hitungPanjang() << endl;
90
91     return 0;
92 }
```

IV. Kesimpulan

Single Linked List adalah struktur data yang terdiri dari node yang saling terhubung, di mana setiap node memiliki data dan penunjuk ke node berikutnya. Operasi dasar meliputi menambah node di depan dan belakang, serta mencetak semua elemen. Linked list memungkinkan kita menghapus node dengan nilai tertentu dan mencari node dengan nilai spesifik. Selain itu, kita bisa menghitung jumlah elemen dalam linked list. Kelebihan utamanya adalah fleksibilitas dalam penambahan dan penghapusan elemen tanpa perlu menggeser elemen lain, menjadikannya cocok untuk situasi data yang berubah-ubah.