

LAPORAN PRAKTIKUM

Modul 5

Single Linked List

Bagian Kedua



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

A.Searching

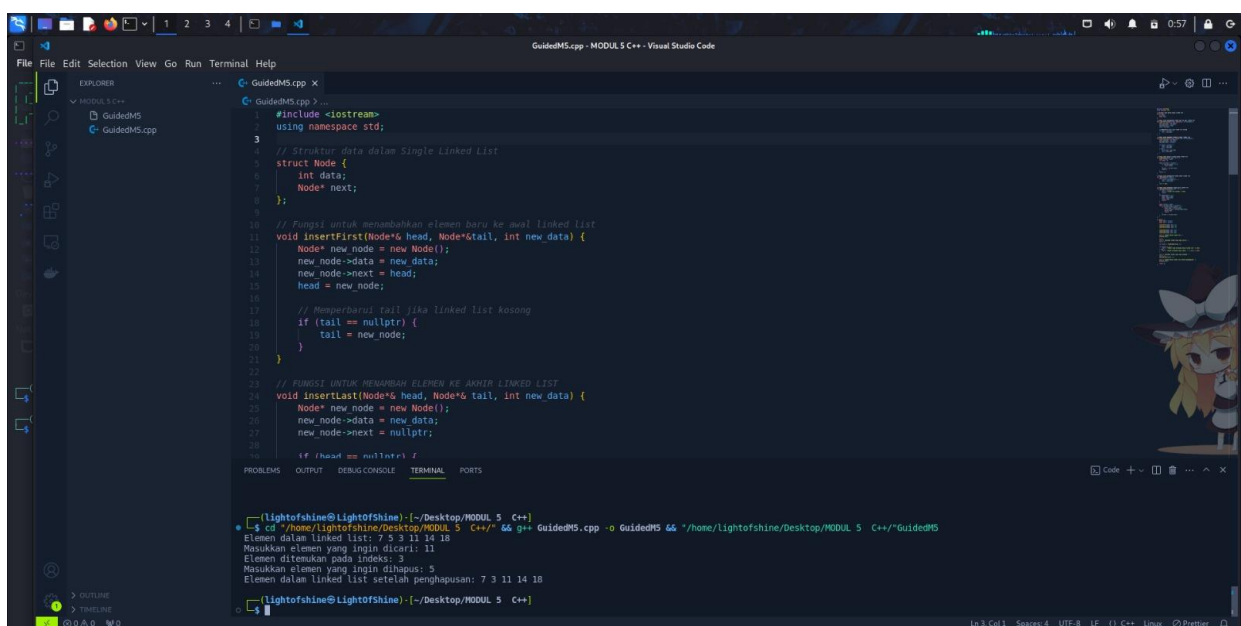
Searching merupakan operasi dasar pada linked list yang dilakukan untuk mencari node tertentu. Proses ini berjalan dengan mengunjungi setiap node satu per satu hingga node yang dicari ditemukan. Dengan melakukan operasi searching, operasi-operasi lainnya seperti insert after, delete after, dan update akan lebih mudah dilaksanakan.

Proses searching dimulai dari node pertama (head) dan terus melanjutkan ke node berikutnya (next) hingga mencapai node yang dicari atau hingga mencapai akhir linked list (nullptr). Jika node yang dicari ditemukan, operasi dapat dilanjutkan sesuai kebutuhan; jika tidak, maka hasil pencarian menunjukkan bahwa node tersebut tidak ada dalam linked list.

Semua fungsi dasar yang terkait dengan operasi searching, serta operasi lainnya, merupakan bagian dari Abstract Data Type (ADT) dari single linked list. Dalam implementasi menggunakan bahasa pemrograman C++, semua ADT tersebut biasanya disimpan dalam file dengan ekstensi .c untuk kode sumber dan .h untuk header file.

III. Guided

File GuidedM5.cpp



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

// Struktur data dalam Single Linked List
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk menambahkan elemen baru ke awal Linked List
void insertFirst(Node*& head, Node*&tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = head;
    head = new_node;
}

// Memperbarui tail jika linked list kosong
if (tail == nullptr) {
    tail = new_node;
}

// FUNGSI UNTUK MENAMBAH ELEMEN KE AKHIR LINKED LIST
void insertLast(Node*& head, Node*&tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = nullptr;
    if (head == nullptr) {
        head = new_node;
    }
}
```

Terminal Output:

```
(LightofShine@LightofShine):[~/Desktop/MODUL 5 C++]
$ cd "/home/lightofshine/Desktop/MODUL 5 C++/" && g++ GuidedM5.cpp -o GuidedM5 && ./GuidedM5
Elemen dalam linked list: 7 5 3 11 14 18
Masukkan elemen yang ingin dicari: 11
Elemen ditemukan pada indeks: 3
Masukkan elemen yang ingin dihapus: 5
Elemen dalam linked list setelah penghapusan: 7 3 11 14 18
```

Untuk Source Codenya Lebih Lengkap Dibawah ini :

```
1 #include <iostream>
2 using namespace std;
3
4 // Struktur data dalam Single Linked List
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 // Fungsi untuk menambahkan elemen baru ke awal linked list
11 void insertFirst(Node*& head, Node*& tail, int new_data) {
12     Node* new_node = new Node();
13     new_node->data = new_data;
14     new_node->next = head;
15     head = new_node;
16
17     // Memperbarui tail jika linked list kosong
18     if (tail == nullptr) {
19         tail = new_node;
20     }
21 }
22
23 // FUNGSI UNTUK MENAMBAH ELEMEN KE AKHIR LINKED LIST
24 void insertLast(Node*& head, Node*& tail, int new_data) {
25     Node* new_node = new Node();
26     new_node->data = new_data;
27     new_node->next = nullptr;
28
29     if (head == nullptr) {
30         head = new_node;
31         tail = new_node;
32     } else {
33         tail->next = new_node;
34         tail = new_node;
35     }
36 }
37
38 // Fungsi untuk mencari elemen dalam linked list
39 int findElement(Node* head, int x) {
40     Node* current = head;
41     int index = 0;
42
43     while (current != nullptr) {
44         if (current->data == x) {
45             return index;
46         }
47         current = current->next;
48         index++;
49     }
50     return -1;
51 }
52
53 // Fungsi untuk menampilkan elemen dalam linked list
54 void display(Node* node) {
55     while (node != nullptr) {
56         cout << node->data << " ";
57         node = node->next;
58     }
59     cout << endl;
60 }
61
62 // Fungsi untuk menghapus elemen dari linked list
63 void deleteElement(Node*& head, int x) {
64     if (head == nullptr) {
65         cout << "Linked list kosong" << endl;
66         return;
67     }
68     if (head->data == x) {
69         Node* temp = head;
70         head = head->next;
71         delete temp;
72         return;
73     }
74
75     Node* current = head;
76     while (current->next != nullptr) {
77         if (current->next->data == x) {
78             Node* temp = current->next;
79             current->next = current->next->next;
80             delete temp;
81             return;
82         }
83         current = current->next;
84     }
85 }
86
87 int main() {
88     Node* head = nullptr;
89     Node* tail = nullptr;
90
91     insertFirst(head, tail, 3);
92     insertFirst(head, tail, 5);
93     insertFirst(head, tail, 7);
94
95     insertLast(head, tail, 11);
96     insertLast(head, tail, 14);
97     insertLast(head, tail, 18);
98
99     cout << "Elemen dalam linked list: ";
100     display(head);
101
102     int x;
103     cout << "Masukkan elemen yang ingin dicari: ";
104     cin >> x;
105
106     int result = findElement(head, x);
107
108     if (result == -1) {
109         cout << "Elemen tidak ditemukan dalam linked list" << endl;
110     } else {
111         cout << "Elemen ditemukan pada indeks: " << result << endl;
112     }
113
114     cout << "Masukkan elemen yang ingin dihapus: ";
115     cin >> x;
116     deleteElement(head, x);
117
118     cout << "Elemen dalam linked list setelah penghapusan: ";
119     display(head);
120
121     return 0;
122 }
```

IV. Unguided

5.1 Latihan

1. Buatlah ADT *Single Linked list* sebagai berikut di dalam file “singlelist.h”:

```
Type infotype : int
Type address : pointer to ElmList

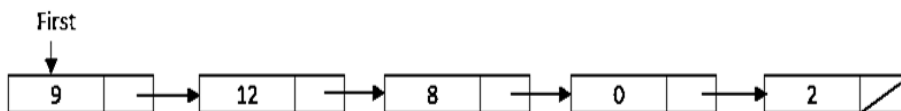
Type ElmList < info :
    infotypenext :
    address
>

Type List : < First : address >

prosedur CreateList( in/out L : List ) fungsi alokasi( x
: infotype ) : addressprosedur dealokasi( in/out P :
address )prosedur printInfo( in L : List )
```

Kemudian buat implementasi ADT *Single Linked list* pada file “singlelist.cpp”.

Adapun isi data



Gambar 5-1 Ilustrasi elemen

Cobalah hasil implementasi ADT pada file “**main.cpp**”

```
int main()
{
    List L;
    address P1, P2, P3, P4, P5 = NULL;
    createList(L);

    P1 = alokasi(2); insertFirst(L,P1);

    P2 = alokasi(0); insertFirst(L,P2);

    P3 = alokasi(8); insertFirst(L,P3);

    P4 = alokasi(12); insertFirst(L,P4);

    P5 = alokasi(9); insertFirst(L,P5);

    printInfo(L);return
    0;
}
```

```
9 12 8 0 2
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Gambar 5-2 Output singelists

- Carilah elemen dengan info 8 dengan membuat fungsi baru.
fungsi findElem(L : List, x : infotype) : address

```
8 ditemukan dalam list
Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

Gambar 5-3 Output pencarian 8

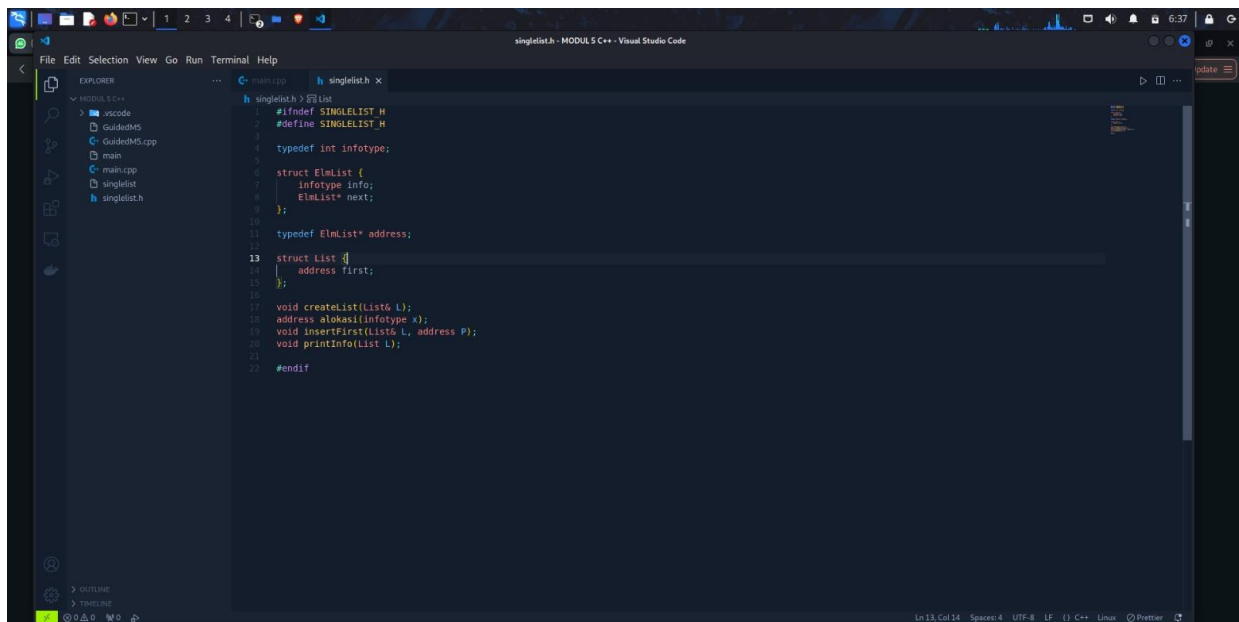
- Hitunglah jumlah total info seluruh elemen (9+12+8+0+2=31).

```
Total info dari kelima elemen adalah 31
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

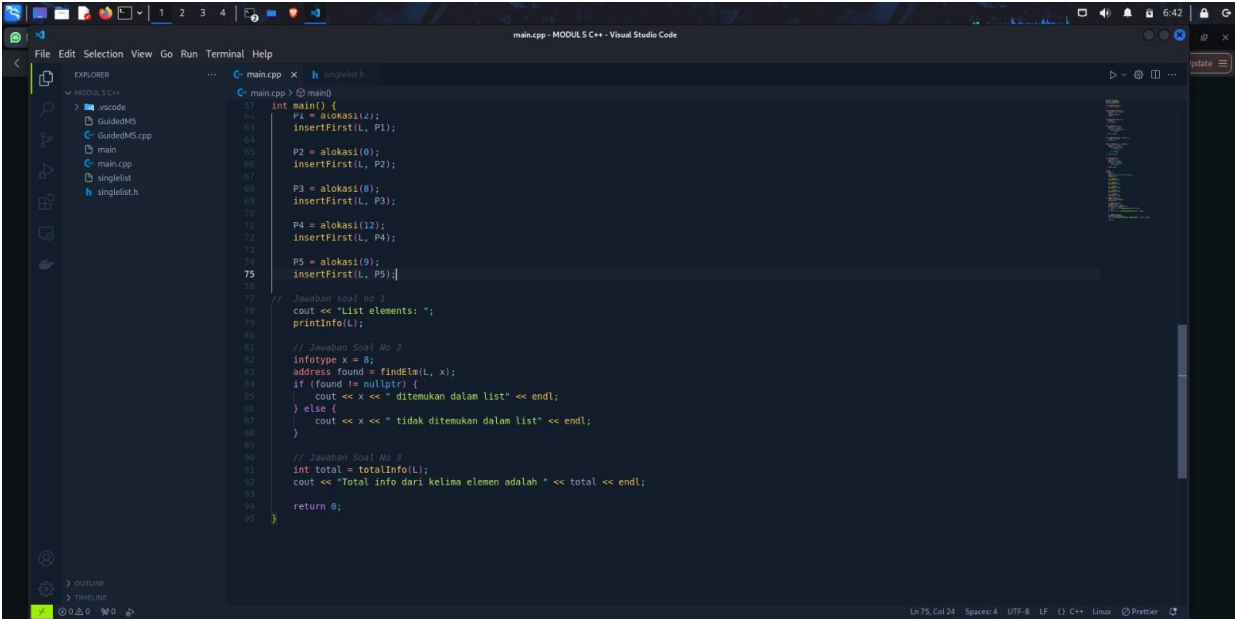
Gambar 5-4 Output total info elemen

Jawab :

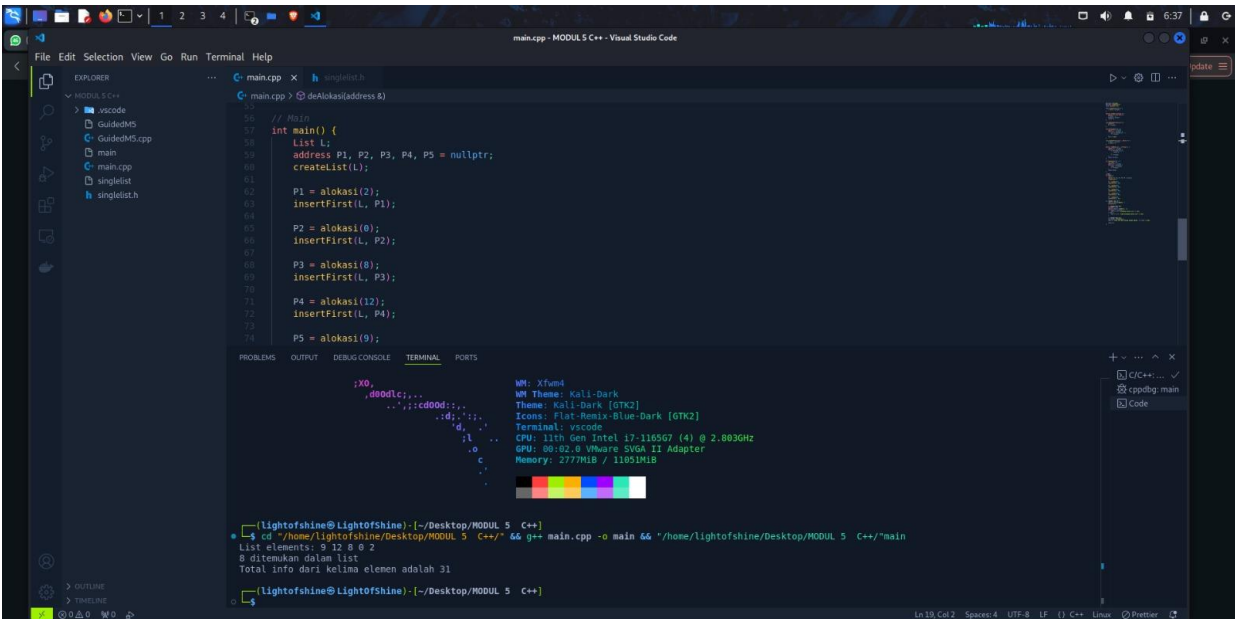
File singlelist.h



File main.cpp



Output :



V. Kesimpulan

Operasi searching dalam single linked list merupakan proses penting untuk menemukan node tertentu dengan mengunjungi setiap node satu per satu hingga mencapai node yang dicari atau akhir dari list. Operasi ini mendukung berbagai fungsi lain, seperti insert after, delete after, dan update, yang bergantung pada pencarian node target.

Dalam implementasi menggunakan bahasa C++, semua operasi dasar single linked list, termasuk pencarian, merupakan bagian dari Abstract Data Type (ADT). Biasanya, fungsi-fungsi ADT ini disimpan dalam file .c sebagai kode sumber dan .h sebagai file header untuk pengelolaan struktur data yang efisien dan modular.

