

**LAPORAN PRAKTIKUM
STRUKTUR DATA
MODUL 4
“SINGLE LINKED LIST (BAGIAN KEDUA) ”**



Disusun Oleh:
Dhiya Ulhaq Ramadhan 2211104053
Kelas :
S1SE-07-02
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
- Mengimplementasikan operasi searching dalam single linked list

2. Landasan Teori

Single Linked List adalah salah satu struktur data linear yang terdiri dari sekumpulan elemen data yang disebut node, dimana setiap node saling terhubung satu sama lain melalui pointer.

Operasi-operasi dasar yang dapat dilakukan pada Single Linked List meliputi pembuatan list kosong (CreateList), penambahan node baru (Insert), penghapusan node (Delete), dan pencarian node (Search). Operasi CreateList digunakan untuk menginisialisasi list kosong dengan pointer first yang menunjuk ke NULL. Operasi Insert dapat dilakukan di awal list (InsertFirst), di akhir list (InsertLast), atau setelah node tertentu (InsertAfter). Operasi Delete juga dapat dilakukan di berbagai posisi seperti awal, akhir, atau setelah node tertentu.

Searching atau pencarian merupakan operasi fundamental dalam Single Linked List yang memungkinkan kita untuk menemukan node dengan nilai tertentu. Proses searching dilakukan dengan mengunjungi setiap node secara sekuensial mulai dari node pertama hingga node yang dicari ditemukan atau mencapai akhir list. Operasi searching menjadi dasar untuk operasi-operasi lain seperti insert after, delete after, dan update, karena sebelum melakukan operasi-operasi tersebut, kita perlu menemukan posisi yang tepat dalam list.

3. Guided

Source code :

```
1  #include <iostream>
2
3  using namespace std;
4
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(); // Perbaiki tipe 'Node' bukan 'node'
13     new_node->data = new_data;
14     new_node->next = head;
15     head = new_node;
16
17     if (tail == nullptr) {
18         tail = new_node;
19     }
20 }
21
22 // Fungsi untuk menambahkan elemen baru ke akhir linked list
23 void insertLast(Node*& head, Node*& tail, int new_data) {
24     Node* new_node = new Node(); // Perbaiki tipe 'Node' bukan 'node'
25     new_node->data = new_data;
26     new_node->next = nullptr;
27
28     if (head == nullptr) {
29         head = new_node;
30         tail = new_node;
31     } else { // 'Else' diganti menjadi 'else'
32         tail->next = new_node;
33         tail = new_node;
34     }
35 }
36
37 // Fungsi untuk mencari elemen di linked list
38 int findElement(Node* head, int x) {
39     Node* current = head;
40     int index = 0;
41
42     while (current != nullptr) {
43         if (current->data == x) {
44             return index;
45         }
46         current = current->next;
47         index++;
48     }
49     return -1;
50 }
51
52 // Fungsi untuk menampilkan elemen dalam linked list
53 void display(Node* node) {
54     while (node != nullptr) {
55         cout << node->data << " ";
56         node = node->next;
57     }
58     cout << endl;
59 }
```

```

61 // Fungsi untuk menghapus elemen dari linked list
62 void deleteElement(Node*& head, int x) {
63     if (head == nullptr) {
64         cout << "Linked list kosong" << endl;
65         return;
66     }
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    cout << "Masukkan elemen yang ingin dihapus: ";
114    cin >> x;
115    deleteElement(head, x);
116
117    cout << "Elemen dalam linked list setelah penghapusan: ";
118    display(head);
119
120    return 0;
121 }

```

Output :

```

D:\bersama berkarya\SEMES  x + v
Elemen dalam linked list: 7 5 3 11 14 18
Masukkan elemen yang ingin dicari: 5
Elemen ditemukan pada indeks 1
Masukkan elemen yang ingin dihapus: 14
Elemen dalam linked list setelah penghapusan: 7 5 3 11 18

Process returned 0 (0x0)   execution time : 7.082 s
Press any key to continue.

```

Deskripsi program

Program memiliki beberapa fungsi utama untuk memanipulasi linked list. Fungsi `insertFirst` digunakan untuk menambahkan elemen baru di awal list dengan cara membuat node baru dan menempatkannya sebagai head. Jika list masih kosong, node tersebut juga akan menjadi tail. Sedangkan fungsi `insertLast` digunakan untuk menambahkan elemen di akhir list dengan menempatkan node baru setelah tail, atau menjadikannya sebagai head dan tail jika list masih kosong.

Untuk pencarian elemen, terdapat fungsi `findElement` yang akan menelusuri list dari awal hingga menemukan nilai yang dicari. Fungsi ini akan mengembalikan posisi (indeks) dari elemen jika ditemukan, atau nilai -1 jika tidak ditemukan. Fungsi `display` digunakan untuk menampilkan seluruh elemen dalam list dengan cara mencetak data dari setiap node secara berurutan dari head hingga tail.

Program juga dilengkapi dengan fungsi `deleteElement` untuk menghapus node dengan nilai tertentu. Fungsi ini memiliki penanganan khusus untuk berbagai kasus seperti list kosong, penghapusan di head, atau penghapusan di tengah/akhir list.

Dalam fungsi `main`, program mendemonstrasikan penggunaan semua fungsi tersebut. Dimulai dengan membuat linked list kosong, kemudian menambahkan beberapa elemen baik di awal maupun di akhir list. Program akan meminta input dari pengguna untuk mencari dan menghapus elemen tertentu, serta menampilkan hasil setiap operasi yang dilakukan.

1.

Buatlah ADT *Single Linked list* sebagai berikut di dalam file "**singlelist.h**":

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

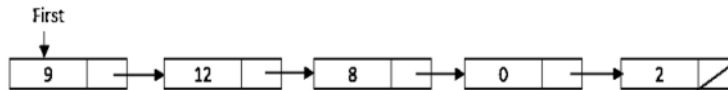
Type Elmlist <
    info : infotype
    next : address
>

Type List : < First : address >

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

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;
}
```

Jawaban

Source code :

Singlelist.cpp

```
1  #include <iostream>
2  #include "singlelist.h"
3  using namespace std;
4
5  void createList(List &L) {
6      L.First = NULL;
7  }
8
9  address alokasi(infotype x) {
10     address P = new ElmList;
11     P->info = x;
12     P->next = NULL;
13     return P;
14 }
15
16 void dealokasi(address &P) {
17     delete P;
18     P = NULL;
19 }
20
21 void printInfo(List L) {
22     address P = L.First;
23     while (P != NULL) {
24         cout << P->info << " ";
25         P = P->next;
26     }
27     cout << endl;
28 }
29
30 void insertFirst(List &L, address P) {
31     P->next = L.First;
32     L.First = P;
33 }
```

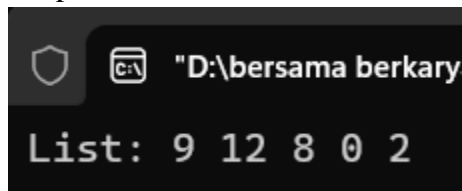
Singlelist.h

```
1  #ifndef SINGLELIST_H
2  #define SINGLELIST_H
3
4  typedef int infotype;
5  typedef struct ElmList *address;
6
7  struct ElmList {
8     infotype info;
9     address next;
10 };
11
12 struct List {
13     address First;
14 };
15
16 // Deklarasi fungsi dan prosedur
17 void createList(List &L);
18 address alokasi(infotype x);
19 void dealokasi(address &P);
20 void printInfo(List L);
21 void insertFirst(List &L, address P);
22
```

Main.cpp

```
1  #include <iostream>
2  #include "singlelist.h"
3  using namespace std;
4
5  int main() {
6      List L;
7      address P1, P2, P3, P4, P5 = NULL;
8
9      createList(L);
10
11     // Membuat linked list
12     P1 = alokasi(2);
13     insertFirst(L, P1);
14
15     P2 = alokasi(0);
16     insertFirst(L, P2);
17
18     P3 = alokasi(8);
19     insertFirst(L, P3);
20
21     P4 = alokasi(12);
22     insertFirst(L, P4);
23
24     P5 = alokasi(9);
25     insertFirst(L, P5);
26
27     cout << "List: ";
28     printInfo(L);
```

Output :



```

D:\bersama berkary
List: 9 12 8 0 2
```

Deskripsi program

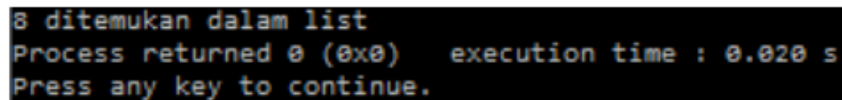
File Singlelist.h berfungsi sebagai header file yang berisi deklarasi tipe data dan prototype fungsi-fungsi yang akan digunakan. Dalam file ini didefinisikan beberapa tipe data penting seperti infotype yang merupakan tipe data integer, address yang merupakan pointer ke struct ElmList, serta dua buah struct yaitu ElmList dan List. Struct ElmList memiliki dua komponen yaitu info untuk menyimpan nilai dan next sebagai pointer ke node berikutnya, sedangkan struct List memiliki satu komponen yaitu first yang merupakan pointer ke node pertama dalam list.

File Singlelist.cpp berisi implementasi dari fungsi-fungsi yang telah dideklarasikan dalam header file. File ini mengimplementasikan fungsi createList untuk membuat list kosong dengan menginisialisasi first menjadi NULL, fungsi alokasi untuk membuat node baru dan mengisinya dengan nilai parameter, fungsi dealokasi untuk menghapus node dari memori, fungsi printInfo untuk menampilkan seluruh isi list, serta fungsi insertFirst untuk menambahkan node baru di awal list.

File Main.cpp merupakan program utama yang mendemonstrasikan penggunaan struktur data Single Linked List. Dalam program ini dibuat sebuah list kosong yang kemudian diisi dengan lima buah node yang masing-masing memiliki nilai 2, 0, 8, 12, dan 9. Proses penambahan node dilakukan menggunakan fungsi insertFirst, sehingga setiap node baru akan menjadi node pertama dalam list. Hal ini menyebabkan urutan nilai dalam list menjadi terbalik dari urutan pemasukan, yaitu 9, 12, 8, 0, 2 seperti yang ditunjukkan pada output program.

2.

Carilah elemen dengan info 8 dengan membuat fungsi baru.
 fungsi findElm(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

Jawaban : Lanjutan dari code pada no 1

Source code :

Singlelist.cpp

```
36 address findElm(List L, infotype x) {
37     address P = L.First;
38     while (P != NULL) {
39         if (P->info == x) {
40             return P;
41         }
42         P = P->next;
43     }
44     return NULL;
45 }
```

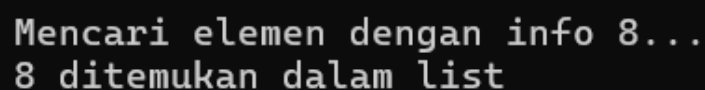
Singlelist.h

```
24 address findElm(List L, infotype x);
```

Main.cpp

```
30 // Mencari elemen dengan info 8
31 cout << "\nMencari elemen dengan info 8..." << endl;
32 address result = findElm(L, 8);
33
34 if (result != NULL) {
35     cout << "8 ditemukan dalam list" << endl;
36 } else {
37     cout << "8 tidak ditemukan dalam list" << endl;
38 }
```

Output :



```
Mencari elemen dengan info 8...
8 ditemukan dalam list
```

Deskripsi program

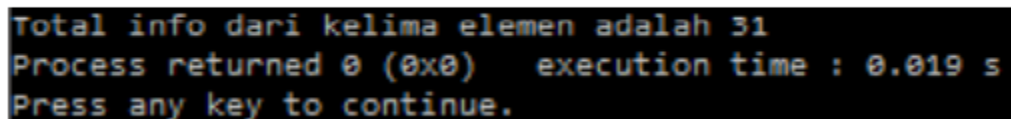
implementasi fungsi pencarian (findElm) dalam Single Linked List yang sebelumnya telah dibuat. Fungsi findElm bertujuan untuk mencari sebuah elemen dalam list berdasarkan nilai informasi yang diberikan. Implementasinya dideklarasikan dalam file header Singlelist.h dan didefinisikan dalam file Singlelist.cpp.

Dalam file Singlelist.cpp, fungsi findElm menggunakan parameter List L yang merupakan list yang akan dicari dan infotype x yang merupakan nilai yang dicari. Fungsi ini melakukan traversal atau penelusuran list dimulai dari elemen pertama (L.First).

Dalam file Main.cpp, fungsi findElm digunakan untuk mencari elemen dengan nilai 8 dalam list yang telah dibuat sebelumnya. Hasil pencarian disimpan dalam variabel result. Program kemudian melakukan pengecekan terhadap nilai result. Jika result tidak NULL (artinya elemen ditemukan), program akan menampilkan pesan "8 ditemukan dalam list". Sebaliknya, jika result bernilai NULL (artinya elemen tidak ditemukan), program akan menampilkan pesan "8 tidak ditemukan dalam list".

3.

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



Jawaban : lanjutan dari code no 1 dan 2

Source code :

Singlelist.cpp

```
48 int sumInfo(List L) {
49     address P = L.First;
50     int total = 0;
51     while (P != NULL) {
52         total += P->info;
53         P = P->next;
54     }
55     return total;
56 }
```

Singlelist.h

```
25 int sumInfo(List L);
```

Main.cpp

```
41 cout << "\nMenghitung total info seluruh elemen..." << endl;  
42 int total = sumInfo(L);  
43 cout << "Total info dari kelima elemen adalah " << total << endl;  
44  
45 return 0;  
46 }
```

Output :

```
Menghitung total info seluruh elemen...  
Total info dari kelima elemen adalah 31
```

Deskripsi program

implementasi fungsi sumInfo yang bertujuan untuk menghitung total nilai informasi dari seluruh elemen dalam Single Linked List. Fungsi ini dideklarasikan dalam file header Singlelist.h dan diimplementasikan dalam file Singlelist.cpp.

Dalam file Singlelist.cpp, fungsi sumInfo menerima parameter List L dan mengembalikan nilai integer yang merupakan total dari seluruh info dalam list. Implementasi fungsi dimulai dengan menginisialisasi pointer P yang menunjuk ke elemen pertama list (L.First) dan variabel total yang diinisialisasi dengan nilai 0. Fungsi kemudian melakukan traversal atau penelusuran list menggunakan perulangan while yang akan berjalan selama pointer P belum mencapai NULL.

Dalam file Main.cpp, fungsi sumInfo dipanggil untuk menghitung total nilai info dari list yang telah dibuat sebelumnya. Hasil perhitungan disimpan dalam variabel total. Program kemudian menampilkan pesan yang memberitahu user bahwa proses penghitungan total info sedang dilakukan, diikuti dengan hasil perhitungan total tersebut. Output program menunjukkan bahwa total info dari kelima elemen dalam list adalah 31.

Kesimpulan

Setelah melakukan praktikum pada siang hari tadi, saya dapat memahami dan mengimplementasikan berbagai operasi dasar linked list seperti pembuatan list, penambahan elemen, pencarian, dan perhitungan total elemen