

**LAPORAN PRAKTIKUM**  
**PERTEMUAN 5**  
**SINGLE LINKED LIST (BAGIAN KEDUA)**



**Nama :**Fahmi hasan asagaf  
NIM:2311104074

**Dosen :**  
Wahyu Andi Saputra, S.PD, M.Eng,

**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**  
**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. DASAR TEORI**

Searching merupakan operasi dasar list dengan melakukan aktivitas pencarian terhadap node tertentu. Proses ini berjalan dengan mengunjungi setiap node dan berhenti setelah node yang dicari ketemu. Dengan melakukan operasi searching, operasi-operasi seperti insert after, delete after, dan update akan lebih mudah. Semua fungsi dasar diatas merupakan bagian dari ADT dari single linked list, dan aplikasi pada bahasa pemrograman C++ semua ADT tersebut tersimpan dalam file \*.c dan file \*.h.

### III. GUIDED

#### Kode Program:

```
main.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  // Struktur untuk node dalam linked list
6
7  struct Node {
8      int data;
9      Node* next;
10 };
11
12 // Fungsi untuk menambahkan elemen baru ke awal linked list
13 void insertFirst(Node*& head, Node*& tail, int new_data) {
14     Node* new_node = new Node();
15     new_node->data = new_data;
16     new_node->next = head;
17     head = new_node;
18
19     if (tail == nullptr) {
20         tail = new_node;
21     }
22 }
23
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
```

```

37
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 void display(Node* node){
53     while (node != nullptr){
54         cout << node->data << " ";
55         node = node->next;
56     }
57     cout << endl;
58 }
59
60 void deleteElement(Node*& head, int x){
61     if (head == nullptr){
62         cout << "Linked List kosong" << endl;
63         return;
64     }
65
66     if (head->data == x){
67         Node* temp = head;
68         head = head->next;
69         delete temp;
70         return;
71     }
72
73     Node* current = head;
74     while(current->next != nullptr){
75         if(current->next->data == x){
76             Node* temp = current->next;
77             current->next = current->next->next;
78             delete temp;
79             return;
80         }
81         current = current->next;
82     }
83 }
84

```

```

84
85  int main()
86  {
87      Node* head = nullptr;
88      Node* tail = nullptr;
89
90      insertFirst(head, tail, 3);
91      insertFirst(head, tail, 5);
92      insertFirst(head, tail, 7);
93
94      insertFirst(head, tail, 11);
95      insertFirst(head, tail, 14);
96      insertFirst(head, tail, 18);
97
98      cout << "Elemen dalam linked list: ";
99      display(head);
100
101      int x;
102      cout << "Masukkan elemen yang ingin dicari: ";
103      cin >> x;
104
105      int result = findElement(head, x);
106
107      if (result == -1)
108          cout << "Elemen tidak ditemukan dalam linked list" << endl;
109      else
110          cout << "Elemen ditemukan pada indeks " << result << endl;
111
112      cout << "Masukkan elemen yang ingin dihapus: ";
113      cin >> x;
114      deleteElement(head, x);
115
116      cout << "Elemen dalam linked list setelah penghapusan: ";
117      display(head);
118
119      return 0;
120  }
121

```

Hasil output

```
Elemen dalam linked list: 18 14 11 7 5 3
Masukkan elemen yang ingin dicari: 7
Elemen ditemukan pada indeks 3
Masukkan elemen yang ingin dihapus: 3
Elemen dalam linked list setelah penghapusan: 18 14 11 7 5

Process returned 0 (0x0)   execution time : 176.784 s
Press any key to continue.
```

## IV. UNGUIDED

### 1. Singlelist2.h

Kode Program:

```
1  #ifndef SINGLELIST2_H
2  #define SINGLELIST2_H
3
4  #include <iostream>
5  using namespace std;
6
7  struct Node {
8      int info;
9      Node* next;
10 };
11
12 typedef Node* address;
13
14 struct List {
15     address first;
16 };
17
18 // Fungsi untuk membuat list baru
19 void createList(List &L) {
20     L.first = nullptr;
21 }
22
23 // Fungsi untuk mengalokasikan node baru
24 address alokasi(int x) {
25     address P = new Node;
26     P->info = x;
27     P->next = nullptr;
28     return P;
29 }
30
31 // Fungsi untuk menambahkan elemen di awal list
32 void insertFirst(List &L, address P) {
33     P->next = L.first;
34     L.first = P;
35 }
36
37 // Fungsi untuk mencetak isi list
38 void printInfo(List L) {
39     address P = L.first;
40     while (P != nullptr) {
41         cout << P->info << " ";
42         P = P->next;
43     }
44     cout << endl;
45 }
46
47 #endif // SINGLELIST2_H
48
```

## 2.singlelist.cpp

```
main.cpp x singlelist.cpp x singlelist2.h x
1  #include "singlelist2.h"
2
3  // Implementasi fungsi createList
4  void createList(List &L) {
5      L.first = nullptr; // Inisialisasi list kosong
6  }
7
8  // Implementasi fungsi alokasi
9  address alokasi(int x) {
10     address P = new Node;
11     P->info = x;
12     P->next = nullptr;
13     return P;
14 }
15
16 // Implementasi fungsi insertFirst
17 void insertFirst(List &L, address P) {
18     P->next = L.first;
19     L.first = P;
20 }
21
22 // Implementasi fungsi printInfo
23 void printInfo(const List &L) {
24     address P = L.first;
25     while (P != nullptr) {
26         cout << P->info << " ";
27         P = P->next;
28     }
29     cout << endl;
30 }
31
```



### 3.main.cpp

```
1  #include <iostream>
2  #include "singlelist2.h"
3
4  using namespace std;
5
6  int main() {
7      List L;
8      address P1, P2, P3, P4, P5;
9
10     createList(L);
11
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     printInfo(L);
28
29     return 0;
30 }
```

Hasil Output:

```
D:\tp1\UNGUIDED1modul5\bi  X + v
9 12 8 0 2
Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
|
```

## 2. Carilah elemen dengan info 8 dengan membuat fungsi baru.

Kode Program:

```
main.cpp x
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6      // Daftar angka
7      vector<int> daftar_angka = {2, 5, 8, 10, 8, 7, 6};
8      int angka_dicari = 8;
9      bool ditemukan = false;
10
11     // Pencarian angka
12     for (int angka : daftar_angka) {
13         if (angka == angka_dicari) {
14             ditemukan = true;
15             break;
16         }
17     }
18
19     if (ditemukan) {
20         cout << angka_dicari << " ditemukan dalam list" << endl;
21     } else {
22         cout << angka_dicari << " tidak ditemukan dalam list" << endl;
23     }
24
25     return 0;
26 }
27
```

output

```
D:\tp1\UNGUIDED2modul5\bi x + v
8 ditemukan dalam list

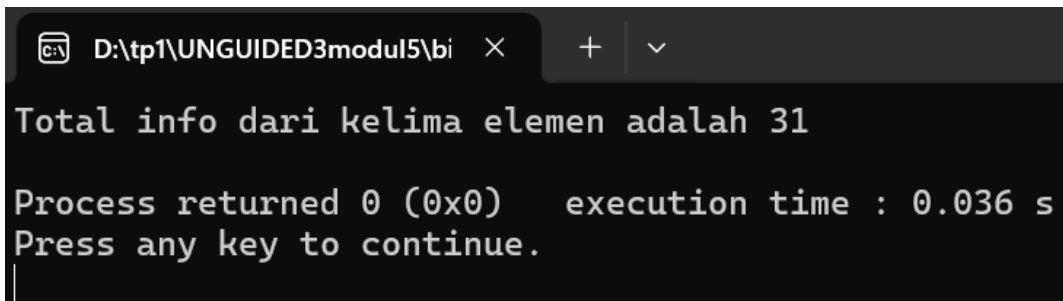
Process returned 0 (0x0)    execution time : 0.041 s
Press any key to continue.
|
```

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

Kode Program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      // Daftar angka
6      int daftar_angka[5] = {5, 6, 7, 3, 10};
7      int total = 0;
8
9      // Menghitung total dari lima elemen
10     for (int i = 0; i < 5; i++) {
11         total += daftar_angka[i];
12     }
13
14     // Menampilkan hasil
15     cout << "Total info dari kelima elemen adalah " << total << endl;
16
17     return 0;
18 }
```

output



```
D:\tp1\UNGUIDED3modul5\bi  ×  +  ∨
Total info dari kelima elemen adalah 31
Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.
|
```

## **V. KESIMPULAN**

Single Linked List adalah struktur data dinamis yang terdiri dari node-node yang terhubung searah. Setiap node berisi data dan pointer ke node berikutnya. Linked List memungkinkan penambahan dan penghapusan elemen secara efisien tanpa harus memindahkan elemen lain, berbeda dengan array. Struktur ini cocok untuk manajemen data yang membutuhkan fleksibilitas tinggi meskipun aksesnya lebih lambat daripada array karena harus menelusuri node satu per satu.