

LAPORAN PRAKTIKUM
Modul 4
SINGLE LINKED LIST (BAGIAN PERTAMA)



Disusun Oleh:

Nama : Ganes Gemi Putra

NIM : 2311104075

Kelas : SE-07-02

Dosen : WAHYU ANDI SAPUTRA

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

Definisi Linked List: Linked list adalah struktur data yang terdiri dari serangkaian elemen data yang saling terkait, di mana setiap elemen mengacu pada elemen berikutnya dalam urutan. Linked list bersifat fleksibel karena ukurannya dapat berubah sesuai kebutuhan.

Keuntungan Menggunakan Pointer :

Pointer digunakan dalam linked list karena sifatnya yang dinamis, lebih efisien dalam hal penambahan atau penghapusan elemen, dan lebih mudah digunakan dibandingkan array yang bersifat statis.

Operasi Dasar pada Linked List :

- Penciptaan dan Inisialisasi List (Create List).
- Penyisipan Elemen (Insert) :

Memasukkan elemen di awal (insert first), di akhir (insert last), atau di tengah list (insert after).

- Penghapusan Elemen (Delete): Menghapus elemen di awal, di akhir, atau di tengah list.
- Penelusuran dan Menampilkan Elemen (View):

Menampilkan semua elemen yang ada dalam linked list.

- Pencarian dan Pengubahan Elemen (Search and Update).

Model-model ADT Linked List yang dibahas meliputi Single Linked List, Double Linked List, Circular Linked List, Multi Linked List, Stack, Queue, Tree, dan Graph.

Implementasi:

Modul ini juga memberikan contoh implementasi struktur data linked list dalam bahasa pemrograman, seperti C dan C++, termasuk bagaimana melakukan alokasi dan dealokasi memori, serta pengecekan apakah list kosong.

Contoh Penggunaan Linked List :

Implementasi untuk kasus data mahasiswa, di mana setiap data terdiri dari nama dan NIM, menunjukkan penggunaan linked list dalam pengelolaan data yang lebih kompleks.

Modul ini bertujuan agar pembaca memahami dasar-dasar penggunaan linked list serta mampu mengimplementasikannya dalam program menggunakan pointer.

Guided

```
Start here X main.cpp X
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // Deklarasi Struct untuk mahasiswa
6 struct mahasiswa {
7     char nama[30];
8     char nim[10];
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 // Pengujian 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     }
39 }
```

```
D:\LAPRAKCPP\modul 4\mai x + - _
Nama: Alice, NIM: 123456
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
List berhasil terhapus!

Process returned 0 (0x0)   execution time : 0.060 s
Press any key to continue.
```

PP C/C++ Windows (CR+LF) WINDOWS-1252 Line 154, Col 2, Pos 3149

```
main.cpp X Guided 2.cpp X
1 #include <iostream>
2 using namespace std;
3
4 // Deklarasi struct untuk elemen list
5 struct Node {
6     int data; // Menyimpan nilai elemen
7     Node* next; // Pointer ke elemen berikutnya
8 };
9
10 // Fungsi untuk mengalokasikan memori untuk node baru
11 Node* alokasi(int value) {
12     Node* newNode = new Node; // Alokasi memori untuk elemen baru
13     if (newNode != nullptr) { // Jika alokasi berhasil
14         newNode->data = value; // Simpan data node
15         newNode->next = nullptr; // Set next ke nullptr
16     }
17     return newNode; // Mengembalikan pointer node baru
18 }
19
20 // Fungsi untuk dealokasi memori node
21 void dealokasi(Node* node) {
22     delete node; // Mengembalikan memori yang digunakan oleh node
23 }
24
25 // Pengujian apakah list kosong
26 bool isEmpty(Node* head) {
27     return head == nullptr; // List kosong jika head adalah nullptr
28 }
29
30 // Menambahkan elemen di awal list
31 void insertFirst(Node* head, int value) {
32     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
33     if (newNode != nullptr) {
34         newNode->next = head; // Menghubungkan elemen baru ke elemen pertama
35         head = newNode; // Menetapkan elemen baru sebagai elemen pertama
36     }
37 }
```

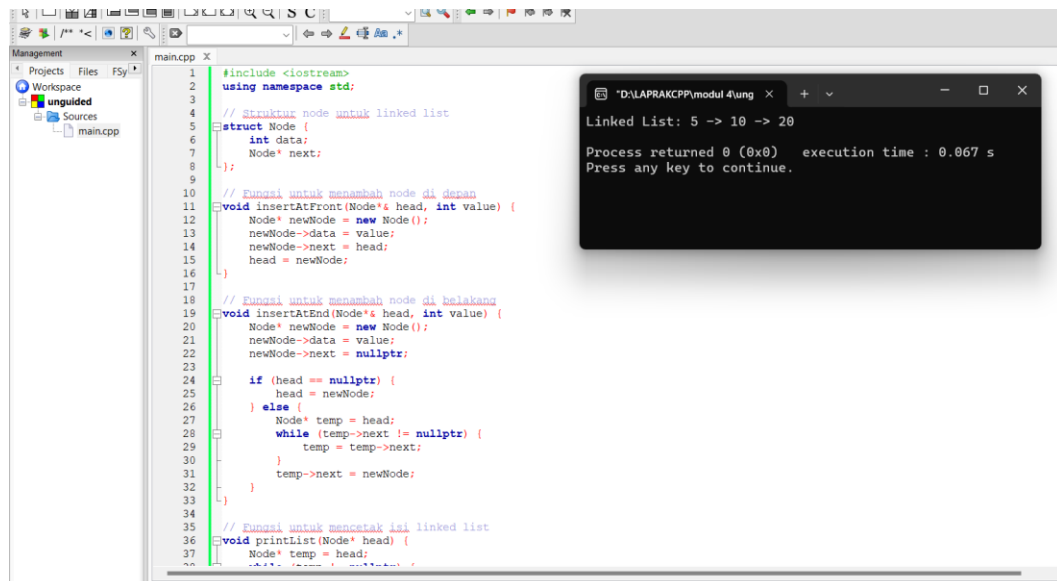
```
D:\LAPRAKCPP\modul 4\Gui x + - _
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.058 s
Press any key to continue.
```

3. Unguided

1. Membuat Single Linked List

Program ini membuat single linked list dengan operasi dasar: menambah node di depan, menambah node di belakang, dan mencetak linked list.



The screenshot shows a C++ IDE with a file named `main.cpp` open. The code implements a single linked list with the following functions:

- `insertAtFront`: Adds a new node at the beginning of the list.
- `insertAtEnd`: Adds a new node at the end of the list.
- `printList`: Prints the contents of the linked list.

The `Node` structure is defined as:

```
struct Node {
    int data;
    Node* next;
};
```

The execution output in the terminal window is:

```
Linked List: 5 -> 10 -> 20
Process returned 0 (0x0)   execution time : 0.067 s
Press any key to continue.
```

2. Menghapus Node pada Linked List

Program ini menghapus node berdasarkan nilai yang diberikan dan kemudian mencetak linked list setelah penghapusan.

```
60
61 */
62
63 #include <iostream>
64 using namespace std;
65
66 // Struktur node untuk linked list
67 struct Node {
68     int data;
69     Node* next;
70 };
71
72 // Fungsi untuk menambah node di depan
73 void insertAtFront(Node*& head, int value) {
74     Node* newNode = new Node();
75     newNode->data = value;
76     newNode->next = head;
77     head = newNode;
78 }
79
80 // Fungsi untuk menambah node di belakang
81 void insertAtEnd(Node*& head, int value) {
82     Node* newNode = new Node();
83     newNode->data = value;
84     newNode->next = nullptr;
85
86     if (head == nullptr) {
87         head = newNode;
88     } else {
89         Node* temp = head;
90         while (temp->next != nullptr) {
91             temp = temp->next;
92         }
93         temp->next = newNode;
94     }
95 }
96
97 // Fungsi untuk menghapus node dengan nilai tertentu
```

Terminal Output:

```
"D:\LAPRAKCPP\modul 4\ung x + v
Linked List after deletion: 5 -> 20
Process returned 0 (0x0) execution time : 0.050 s
Press any key to continue.
```

3. Mencari dan Menghitung Panjang Linked List

Program ini mencari node dengan nilai tertentu dan menghitung panjang linked list.

```
63 #include <iostream>
64 using namespace std;
65
66 // Struktur node untuk linked list
67 struct Node {
68     int data;
69     Node* next;
70 };
71
72 // Fungsi untuk menambah node di depan
73 void insertAtFront(Node*& head, int value) {
74     Node* newNode = new Node();
75     newNode->data = value;
76     newNode->next = head;
77     head = newNode;
78 }
79
80 // Fungsi untuk menambah node di belakang
81 void insertAtEnd(Node*& head, int value) {
82     Node* newNode = new Node();
83     newNode->data = value;
84     newNode->next = nullptr;
85
86     if (head == nullptr) {
87         head = newNode;
88     } else {
89         Node* temp = head;
90         while (temp->next != nullptr) {
91             temp = temp->next;
92         }
93         temp->next = newNode;
94     }
95 }
96
97 // Fungsi untuk mencari nilai dalam linked list
98 bool search(Node* head, int value) {
99     Node* temp = head;
100     while (temp != nullptr) {
101         if (temp->data == value) {
102             return true;
103         }
104         temp = temp->next;
105     }
106     return false;
107 }
```

Terminal Output:

```
"D:\LAPRAKCPP\modul 4\ung x + v
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
Process returned 0 (0x0) execution time : 0.054 s
Press any key to continue.
```

4. Kesimpulan

- **Single Linked List** merupakan salah satu struktur data dinamis yang memungkinkan penyimpanan dan pengelolaan data secara efisien. Elemen-elemen dalam linked list saling terhubung melalui pointer, yang membuatnya fleksibel untuk melakukan operasi penambahan dan penghapusan data.
- Penggunaan **pointer** sangat penting dalam linked list karena memungkinkan ukuran list dapat berubah-ubah sesuai kebutuhan, tidak seperti array yang bersifat statis. Pointer juga mempermudah pengelolaan elemen-elemen list, terutama ketika dilakukan penyisipan atau penghapusan data.
- **Operasi-operasi dasar** pada linked list, seperti penciptaan, penyisipan, penghapusan, penelusuran, dan pengubahan data, memberikan fleksibilitas dalam mengelola data secara dinamis. Ini membuat linked list sangat cocok untuk aplikasi yang membutuhkan manipulasi data yang sering.
- Single Linked List hanya dapat diakses secara **sekuensial**, yang berarti bahwa setiap elemen harus diakses secara berurutan. Ini bisa menjadi keterbatasan jika dibandingkan dengan struktur data lain seperti array yang memungkinkan akses langsung.
- Linked list lebih cocok digunakan dalam aplikasi di mana ukuran data tidak diketahui sebelumnya atau sering berubah. Ini termasuk situasi di mana data perlu sering ditambahkan atau dihapus, seperti dalam implementasi stack, queue, dan berbagai algoritma graf.