

LAPORAN PRAKTIKUM

Modul 4

SINGLE LINKED LIST (BAGIAN PERTAMA)



Disusun Oleh:

Rifqi M. Ramdani - 2311104044

SE-07-02

Dosen :

Wahyu Andi Saputra, S.PD, M.Eng,

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. 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

2. Landasan Teori

Linked list adalah salah satu bentuk struktur data berupa serangkaian elemen data yang saling terhubung dan fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Tipe data ini terdiri dari dua jenis: data tunggal dan data majemuk. Penggunaan pointer dalam linked list membuatnya lebih dinamis dan lebih mudah untuk penambahan atau penghapusan elemen dibandingkan array yang bersifat statis.

3. Guided

Linked List dengan Pointer

Linked list (biasa disebut list saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam Linked list bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe string. Sedangkan data majemuk merupakan sekumpulan data (record) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe string, NIM bertipe long integer, dan Alamat bertipe string

Linked list dapat diimplementasikan menggunakan Array dan Pointer (Linked list). Yang akan kita gunakan adalah pointer, karena beberapa alasan, yaitu :

1. Array bersifat statis, sedangkan pointer dinamis.
2. Pada linked list bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai pointer.
3. Sifat linked list yang fleksibel lebih cocok dengan sifat pointer yang dapat diatur sesuai kebutuhan.
4. Karena array lebih susah dalam menangani linked list, sedangkan pointer lebih mudah.
5. Array lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.

Dalam implementasinya, pengaksesan elemen pada Linked list dengan pointer bisa menggunakan (->) atau tanda titik (.). Model-model dari ADT Linked list yang kita pelajari adalah :

1. Single Linked list
2. Double Linked list
3. Circular Linked list
4. Multi Linked list
5. Stack (Tumpukan)
6. Queue (Antrian)
7. Tree
8. Graph

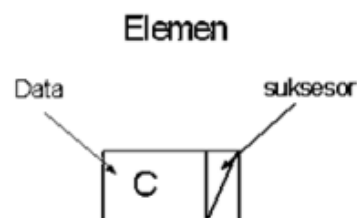
Setiap model ADT Linked list di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan. Secara umum operasi-operasi ADT pada Linked list, yaitu :

1. Penciptaan dan inisialisasi list (Create List).
2. Penyisipan elemen list (Insert).
3. Penghapusan elemen list (Delete).
4. Penelusuran elemen list dan menampilkannya (View).
5. Pencarian elemen list (Searching).
6. Pengubahan isi elemen list (Update)

Single Linked List

Single Linked list merupakan model ADT *Linked list* yang hanya memiliki satu arah *pointer*.

Komponen elemen dalam *single linked list*:



Gambar 4-1 Elemen *Single Linked list*

Keterangan:

Elemen: segmen-segmen data yang terdapat dalam suatu *list*.

Data: informasi utama yang tersimpan dalam sebuah elemen.

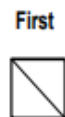
Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari Single Linked list:

1. Hanya memerlukan satu buah pointer.
 2. Node akhir menunjuk ke Nil kecuali untuk list circular.
 3. Hanya dapat melakukan pembacaan maju.
 4. Pencarian sequensial dilakukan jika data tidak terurut.
 5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah list.
- Istilah-istilah dalam Single Linked list :

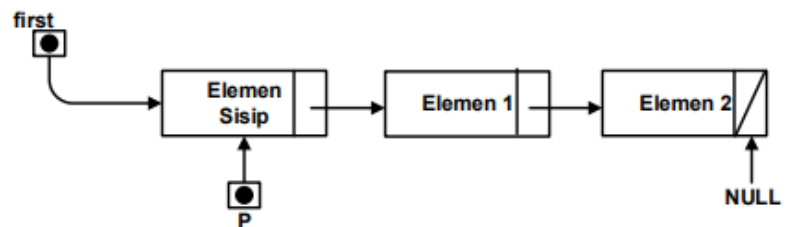
1. first/head: pointer pada list yang menunjuk alamat elemen pertama list.
2. next: pointer pada elemen yang berfungsi sebagai successor (penunjuk) alamat elemen di depannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. Node/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Gambaran sederhana *single linked list* dengan elemen kosong:



Gambar 4-2 *Single Linked list* dengan Elemen Kosong

Gambaran sederhana *single linked list* dengan 3 elemen:



Gambar 4-3 *Single Linked list* dengan 3 Elemen

Contoh deklarasi struktur data *single linked list*:

```
1  /*file : list.h*/
2  #ifndef LIST_H_INCLUDED
3  #define LIST_H_INCLUDED
4
5  #define Nil NULL
6  #define info(P) (P)->info
7  #define next(P) (P)->next
8  #define first(L) ((L).first)
9  using namespace std;
10 /*deklarasi record dan struktur data list*/
11 typedef int infotype;
12 typedef struct elmlist *address;
13 struct elmlist {
14     infotype info;
15     address next;
16 };
17
18 struct list{
19     address first;
20 };
21 #endif // TEST_H_INCLUDED
```

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
1  /*file : list.h*/
2  #ifndef LIST_H_INCLUDED
3  #define LIST_H_INCLUDED
4
5  #define Nil NULL
6  #define info(P) (P)->info
7  #define next(P) (P)->next
8  #define first(L) ((L).first)
9
10 using namespace std;
11 /*deklarasi record dan struktur data list*/
12 struct mahasiswa{
13     char nama[30]
14     char nim[10]
15 }
16 typedef mahasiswa infotype;
17
18 typedef struct elmlist *address;
19 struct elmlist {
20     infotype info;
21     address next;
22 };
23
24 struct list{
25     address first;
26 };
27 #endif // TEST_H_INCLUDED
```

Pembentukan Komponen-Komponen List

A. Pembentukan List A

adalah sebuah proses untuk membuat sebuah list baru. Biasanya nama fungsi yang digunakan `createList()`. Fungsi ini akan mengeset nilai awal list yaitu `first(list)` dan `last(list)` dengan nilai `Nil`. .

B. Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam list. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan `alokasi()`

Sintak alokasi pada C:

```
P = (address) malloc ( sizeof (elmlist));
```

Keterangan:

P = variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address = tipe data *pointer* dari tipe data elemen yang akan dialokasikan.
Elmlist = tipe data atau *record* elemen yang dialokasikan.

Contoh deklarasi struktur data *single linked list*:

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
    address p = (address)malloc(sizeof(elmlist));
    info(p) = m;
    return p;
}
```

Namun pada Cpp. Penggunaan **malloc** dapat dipersingkat menggunakan sintak **new**.

Sintak alokasi pada Cpp:

```
P = new elmlist;
```

Keterangan:

P = variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address = tipe data *pointer* dari tipe data elemen yang akan dialokasikan.

Contoh deklarasi struktur data *single linked list*:

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
    address p = new elmlist;
    info(p) = m;
    return p;
}
```

Dealokasi

Untuk menghapus sebuah memory address yang tersimpan atau telah dialokasikan dalam bahasa pemrograman C digunakan sintak **free**, sedangkan pada Cpp digunakan sintak **delete**, seperti berikut

Sintak pada C:

```
free( p );
```

Sintak pada Cpp:

```
delete p;
```

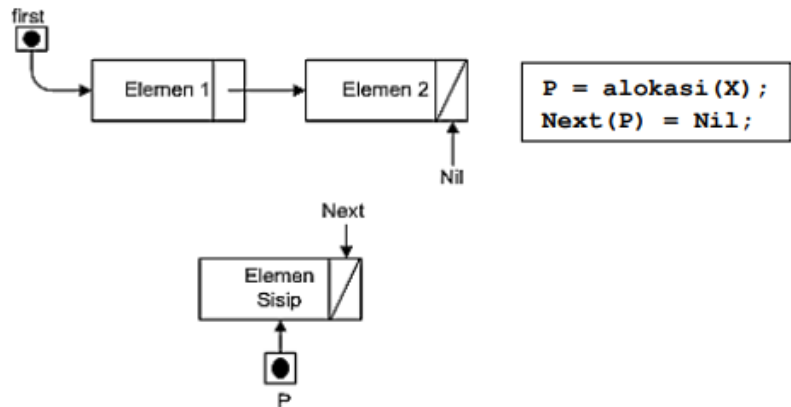
Pengecekan List

Adalah fungsi untuk mengecek apakah list tersebut kosong atau tidak. Akan mengembalikan nilai **true** jika list kosong dan nilai **false** jika list tidak kosong. Fungsi yang digunakan adalah **isEmpty()**.

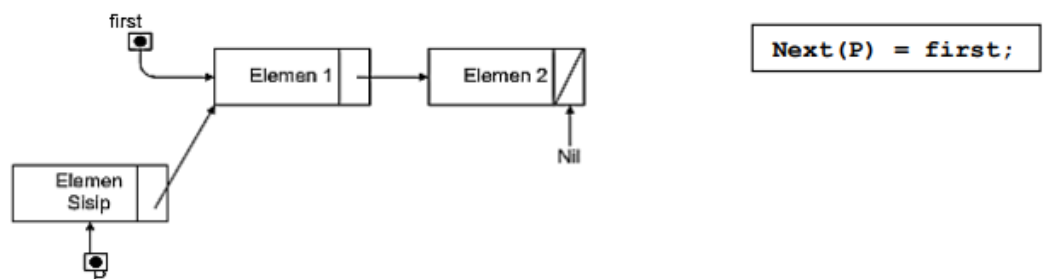
4.2.2 Insert A. I

Insert First

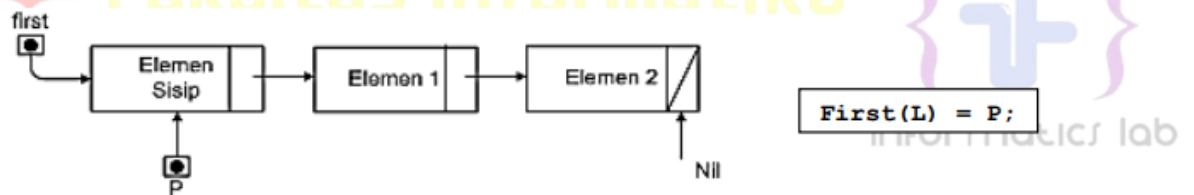
Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada awal list. Langkah-langkah dalam proses insert first:



Gambar 4-4 Single Linked list Insert First (1)



Gambar 4-5 Single Linked list Insert First (2)

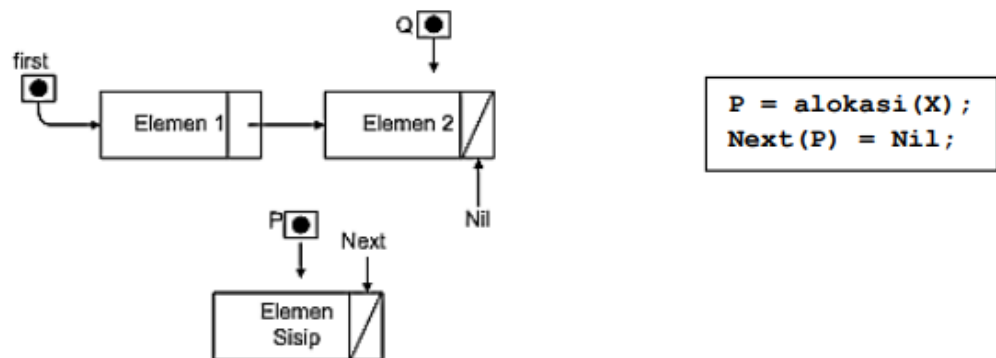


Gambar 4-6 Single Linked list Insert First (3)

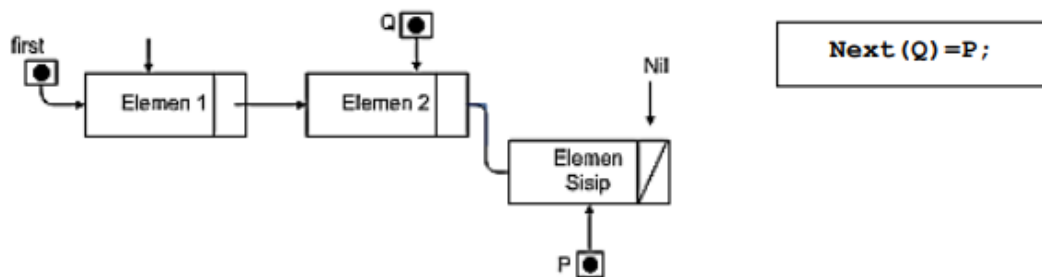
```
/* contoh syntax insert first */
void insertFirst(List &L, address &P){
    next (P) = first(L);
    first(L) = P;
}
```

. Insert Last

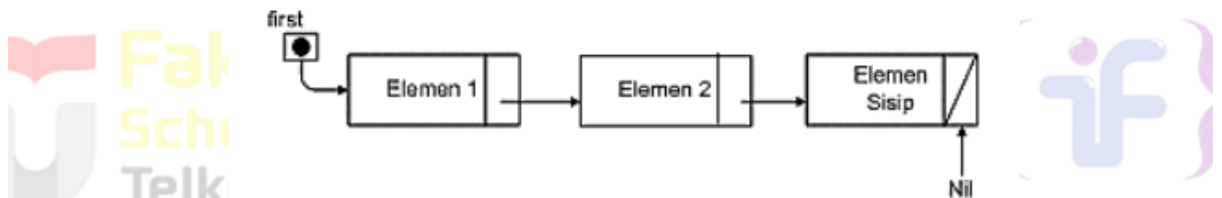
Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada akhir list. Langkah dalam insert last :



Gambar 4-7 Single Linked list Insert Last 1

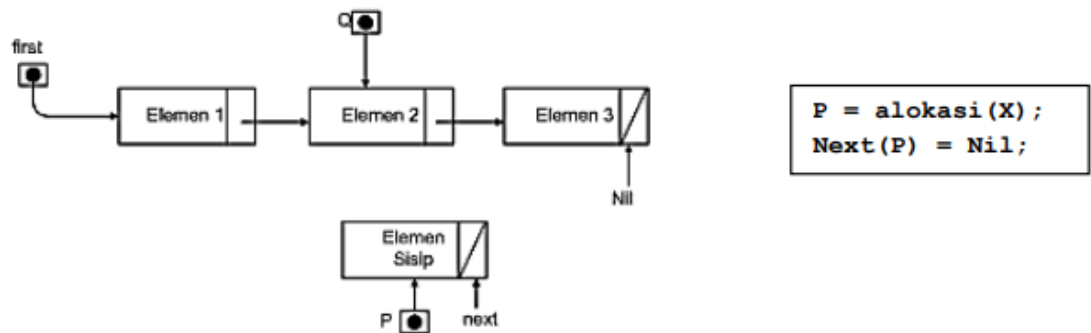


Gambar 4-8 Single Linked list Insert Last 2

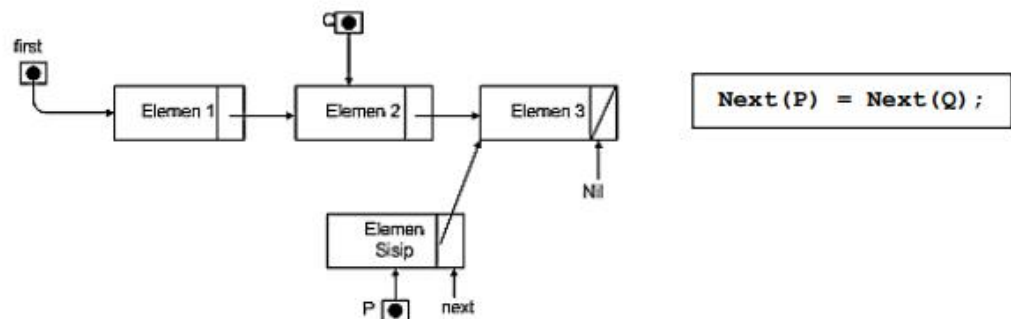


Insert After

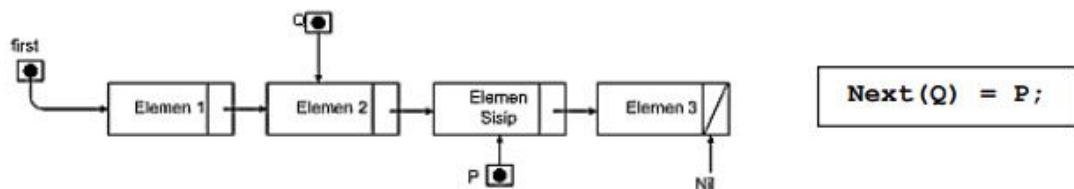
Merupakan metode memasukkan data ke dalam list yang diletakkan setelah node tertentu yang ditunjuk oleh user. Langkah dalam insert after:



Gambar 4-10 Single Linked list Insert After 1



Gambar 4-11 Single Linked list Insert After 2



Gambar 4-12 Single Linked list Insert After 3

View

Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut. Semua fungsi dasar diatas merupakan bagian dari ADT dari single linked list, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam file *.c dan file *.h.

```

1  /*file : list .h*/
2  /* contoh ADT list berkait dengan representasi fisik pointer*/
3  /* representasi address dengan pointer*/
4  /* info tipe adalah integer */
5  #ifndef list_H
6  #define list_H
7  #include "boolean.h"
8  #include <stdio.h>
9  #define Nil NULL
10 #define info(P) (P)->info
11 #define next(P) (P)->next
12 #define first(L) ((L).first)
13
14 /*deklarasi record dan struktur data list*/
15 typedef int infotype;
16 typedef struct elmlist *address;
17 struct elmlist{
18     infotype info;
19     address next;
20 };
21
22 /* definisi list : */
23 /* list kosong jika First(L)=Nil */
24 /* setiap elemen address P dapat diacu info(P) atau next(P) */
25 struct list {
26     address first;
27 };
28 /****** pengecekan apakah list kosong *****/
29 boolean ListEmpty(list L);
30 /*mengembalikan nilai true jika list kosong*/
31
32 /****** pembuatan list kosong *****/
33 void CreateList(list &L);

```

```

34 /* I.S. sembarang
35     F.S. terbentuk list kosong*/
36
37 /****** manajemen memori *****/
38 void dealokasi(address P);
39 /* I.S. P terdefinisi
40     F.S. memori yang digunakan P dikembalikan ke sistem */
41
42 /****** penambahan elemen *****/
43 void insertFirst(list &L, address P);
44 /* I.S. sembarang, P sudah dialokasikan
45     F.S. menempatkan elemen beralamat P pada awal list */
46
47 void insertAfter(list &L, address P, address Prec);
48 /* I.S. sembarang, P dan Prec alamat salah satu elemen list
49     F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
50
51 void insertLast(list &L, address P);
52 /* I.S. sembarang, P sudah dialokasikan
53     F.S. menempatkan elemen beralamat P pada akhir list */
54
55 /****** proses semau elemen list *****/
56 void printInfo(list L);
57 /* I.S. list mungkin kosong
58     F.S. jika list tidak kosong menampilkan semua info yang ada pada list */
59
60 int nbList(list L);
61 /* mengembalikan jumlah elemen pada list */
62
63 #endif

```

Latihan

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

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

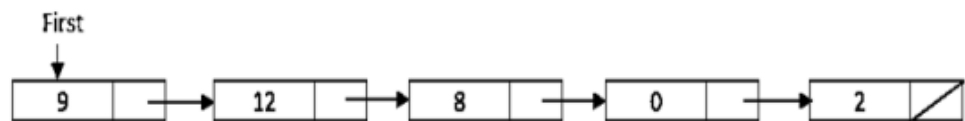
Type ElmList <
    info : infotype
    next : address
>

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 4-13 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;
}
```

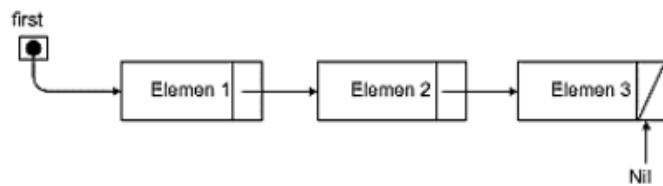
```
9 12 8 0 2
```

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

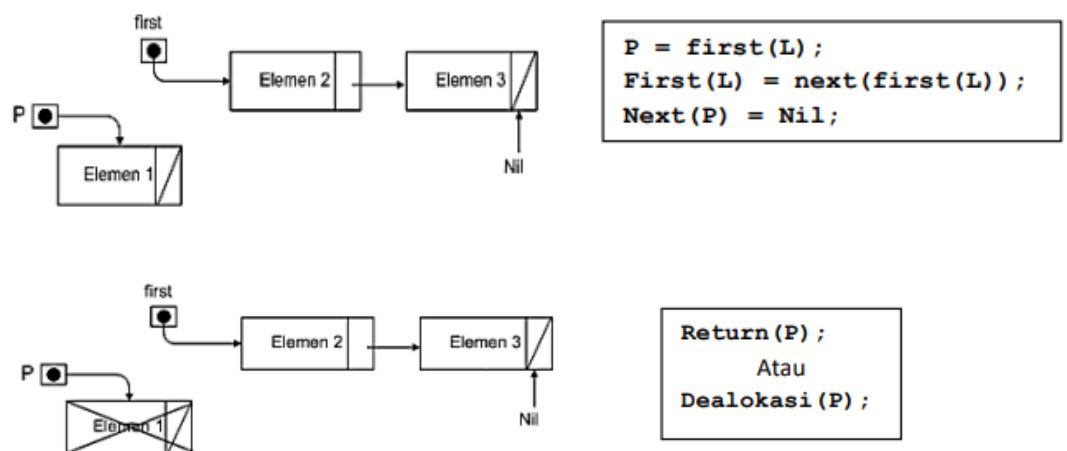
Delete

A. Delete First

Adalah pengambilan atau penghapusan sebuah elemen pada awal list. Langkah-langkah dalam delete first:



Gambar 4-15 Single Linked List Delete First 1



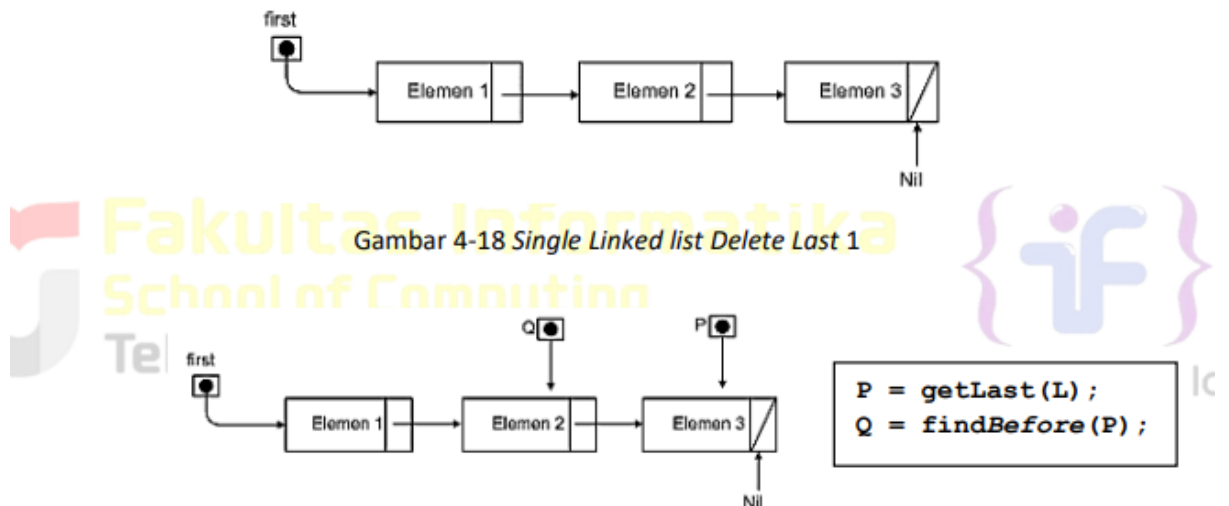
Gambar 4-17 Single Linked list Delete First 3

```

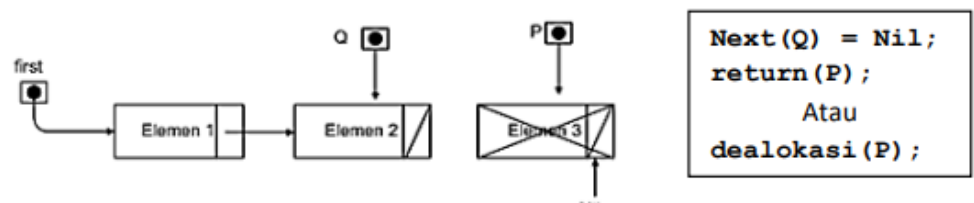
/* contoh syntax delete first */
void deleteFirst(List &L, address &P){
    P = first(L);
    first(L) = next(first(L));
    next (P) = null;
}
  
```

Delete Last

Merupakan pengambilan atau penghapusan suatu elemen dari akhir list. Langkah-langkah dalam delete last:

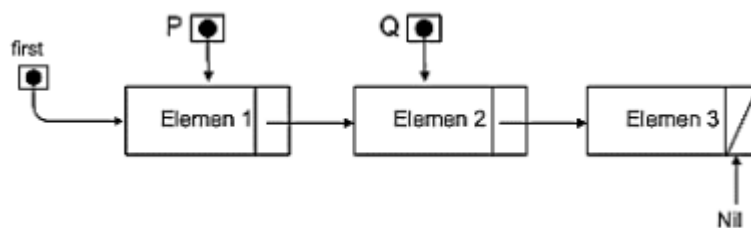


Gambar 4-19 Single Linked list Delete Last 2

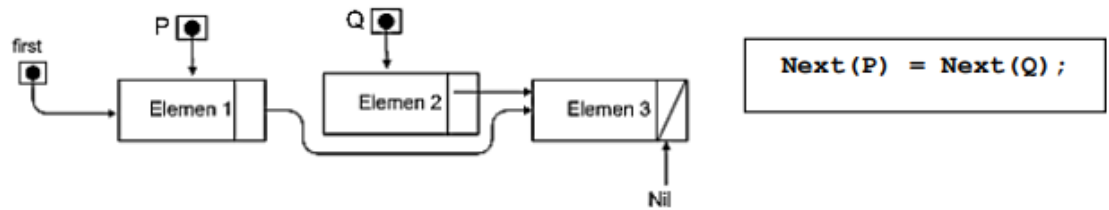


Delete After

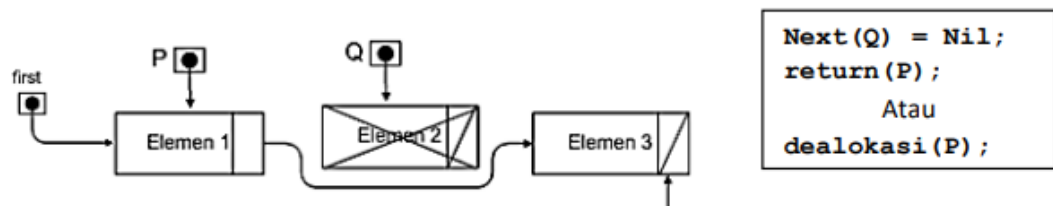
Merupakan pengambilan atau penghapusan node setelah node tertentu. Langkah-langkah dalam delete after:



Gambar 4-21 Single Linked list Delete After 1



Gambar 4-22 Single Linked list Delete After 2



Delete Elemen

Adalah operasi yang digunakan untuk menghapus dan membebaskan memori yang dipakai oleh elemen tersebut. Fungsi yang biasanya dipakai:

1. fungsi dealokasi(P) : membebaskan memori yang dipakai oleh elemen P.
2. fungsi delAll(L) : membebaskan semua memori yang dipakai elemen – elemen yang ada pada list L. Hasil akhir list L menjadi kosong.

Semua operasi-operasi dasar list biasa disebut dengan operasi primitif. Primitif-primitif dalam list ini merupakan bagian dari ADT list yang tersimpan dalam file *.h dan file *.cpp, dengan rincian file *.h untuk menyimpan prototipe primitif-primitif atau fungsi-fungsi dan menyimpan tipe data yang dipergunakan dalam primitif list tersebut.

Untuk bisa mengakses semua primitif tersebut yaitu dengan meng-include terhadap file *.h-nya.

4.5 Update

Merupakan operasi dasar pada list yang digunakan untuk mengupdate data yang ada di dalam list. Dengan operasi update ini kita dapat meng-update data-data node yang ada di dalam list. Proses update biasanya diawali dengan proses pencarian terhadap data yang akan di-update.

LATIHAN GUIDED

Kode Program:

1.

```
main.cpp x
1  #include <iostream>
2  #include <cstring>
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 // 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     strcpy(baru->data.nama, data.nama); // Menggunakan strcpy untuk string
35     strcpy(baru->data.nim, data.nim);   // Menggunakan strcpy untuk string
36     baru->next = nullptr;
37     if (isEmpty()) {
38         head = tail = baru;
39     } else {
```

```

main.cpp X
39     } else {
40         baru->next = head;
41         head = baru;
42     }
43 }
44
45 // Tambah Belakang
46 void insertBelakang(const mahasiswa &data) {
47     Node *baru = new Node;
48     strcpy(baru->data.nama, data.nama); // Menggunakan strcpy untuk string
49     strcpy(baru->data.nim, data.nim);   // Menggunakan strcpy untuk string
50     baru->next = nullptr;
51     if (isEmpty()) {
52         head = tail = baru;
53     } else {
54         tail->next = baru;
55         tail = baru;
56     }
57 }
58
59 // Hitung Jumlah List
60 int hitungList() {
61     Node *current = head;
62     int jumlah = 0;
63     while (current != nullptr) {
64         jumlah++;
65         current = current->next;
66     }
67     return jumlah;
68 }
69
70 // Hapus Depan
71 void hapusDepan() {
72     if (!isEmpty()) {
73         Node *hapus = head;
74         head = head->next;
75         delete hapus;
76         if (head == nullptr) {
77             tail = nullptr; // Jika list menjadi kosong

```

```

main.cpp X
77         tail = nullptr; // Jika list menjadi kosong
78     }
79     } else {
80         cout << "List kosong!" << endl;
81     }
82 }
83
84 // Hapus Belakang
85 void hapusBelakang() {
86     if (!isEmpty()) {
87         if (head == tail) {
88             delete head;
89             head = tail = nullptr; // List menjadi kosong
90         } else {
91             Node *bantu = head;
92             while (bantu->next != tail) {
93                 bantu = bantu->next;
94             }
95             delete tail;
96             tail = bantu;
97             tail->next = nullptr;
98         }
99     } else {
100         cout << "List kosong!" << endl;
101     }
102 }
103
104 // Tampilkan List
105 void tampil() {
106     Node *current = head;
107     if (!isEmpty()) {
108         while (current != nullptr) {
109             cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
110             current = current->next;
111         }
112     } else {
113         cout << "List masih kosong!" << endl;
114     }
115 }

```



```
main.cpp x
115 }
116
117 // Hapus List
118 void clearList() {
119     Node *current = head;
120     while (current != nullptr) {
121         Node *hapus = current;
122         current = current->next;
123         delete hapus;
124     }
125     head = tail = nullptr;
126     cout << "List berhasil terhapus!" << endl;
127 }
128
129 // Main function
130 int main() {
131     init();
132
133     // Contoh data mahasiswa
134     mahasiswa m1 = {"Alice", "123456"};
135     mahasiswa m2 = {"Bob", "654321"};
136     mahasiswa m3 = {"Charlie", "112233"};
137
138     // Menambahkan mahasiswa ke dalam list
139     insertDepan(m1);
140     tampil();
141     insertBelakang(m2);
142     tampil();
143     insertDepan(m3);
144     tampil();
145
146     // Menghapus elemen dari list
147     hapusDepan();
148     tampil();
149     hapusBelakang();
150     tampil();
151
152     // Menghapus seluruh list
153     clearList();
154
155     return 0; // Menggunakan return 0 dengan benar
156 }
157
158
```

Maka Akan Menghasilkan Output

```
"D:\TUGAS SEMESTER 3\Guidi" × + ▾
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 : 0.141 s
Press any key to continue.
|
```

2.

Kode Program

main.cpp X

```
1  #include <iostream>
2  using namespace std;
3
4  // Definisi struktur 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; // Mengisi 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 // Pengecekan apakah list kosong
26 bool isListEmpty(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 }
38
39 // Menambahkan elemen di akhir list
```

main.cpp x

```
39 // Menambahkan elemen di akhir list
40 void insertLast(Node* &head, int value) {
41     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
42     if (newNode != nullptr) {
43         if (isListEmpty(head)) { // Jika list kosong
44             head = newNode;      // Elemen baru menjadi elemen pertama
45         } else {
46             Node* temp = head;
47             while (temp->next != nullptr) { // Mencari elemen terakhir
48                 temp = temp->next;
49             }
50             temp->next = newNode; // Menambahkan elemen baru di akhir list
51         }
52     }
53 }
54
55 // Menampilkan semua elemen dalam list
56 void printList(Node* head) {
57     if (isListEmpty(head)) {
58         cout << "List kosong!" << endl;
59     } else {
60         Node* temp = head;
61         while (temp != nullptr) { // Selama belum mencapai akhir list
62             cout << temp->data << " "; // Menampilkan data elemen
63             temp = temp->next; // Melanjutkan ke elemen berikutnya
64         }
65         cout << endl;
66     }
67 }
68
69 // Menghitung jumlah elemen dalam list
70 int countElements(Node* head) {
71     int count = 0;
72     Node* temp = head;
73     while (temp != nullptr) {
74         count++; // Menambah jumlah elemen
75         temp = temp->next; // Melanjutkan ke elemen berikutnya
76     }
77     return count; // Mengembalikan jumlah elemen
}
```

```

main.cpp X
75     temp = temp->next; // Melanjutkan ke elemen berikutnya
76 }
77 return count;        // Mengembalikan jumlah elemen
78 }
79
80 // Menghapus semua elemen dalam list dan dealokasi memori
81 void clearList(Node* &head) {
82     while (head != nullptr) {
83         Node* temp = head; // Simpan pointer ke node saat ini
84         head = head->next; // Pindahkan ke node berikutnya
85         dealokasi(temp);  // Dealokasi node
86     }
87 }
88
89 int main() {
90     Node* head = nullptr; // Membuat list kosong
91
92     // Menambahkan elemen ke dalam list
93     insertFirst(head, 10); // Menambahkan elemen 10 di awal list
94     insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
95     insertLast(head, 30);  // Menambahkan elemen 30 di akhir list
96
97     // Menampilkan isi list
98     cout << "Isi List: ";
99     printList(head);
100
101     // Menampilkan jumlah elemen
102     cout << "Jumlah elemen: " << countElements(head) << endl;
103
104     // Menghapus semua elemen dalam list
105     clearList(head);
106
107     // Menampilkan isi list setelah penghapusan
108     cout << "Isi List setelah penghapusan: ";
109     printList(head);
110
111     return 0; // Pastikan menggunakan return 0 dengan benar
112 }
113

```

Maka Akan Menghasilkan Output

```

D:\TUGAS SEMESTER 3\Guida X + v
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.101 s
Press any key to continue.
|

```

4. 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

Kode Program:

```

main.cpp x main.cpp x
1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  void insertDepan(Node*& head, int nilai) {
10     Node* newNode = new Node();
11     newNode->data = nilai;
12     newNode->next = head;
13     head = newNode;
14 }
15
16 void insertBelakang(Node*& head, int nilai) {
17     Node* newNode = new Node();
18     newNode->data = nilai;
19     newNode->next = nullptr;
20
21     if (head == nullptr) {
22         head = newNode;
23     } else {
24         Node* temp = head;
25         while (temp->next != nullptr) {
26             temp = temp->next;
27         }
28         temp->next = newNode;
29     }
30 }
31
32 void cetakLinkedList(Node* head) {
33     Node* temp = head;
34     while (temp != nullptr) {
35         cout << temp->data;
36         temp = temp->next;
37         if (temp != nullptr)
38             cout << " -> ";
39     }

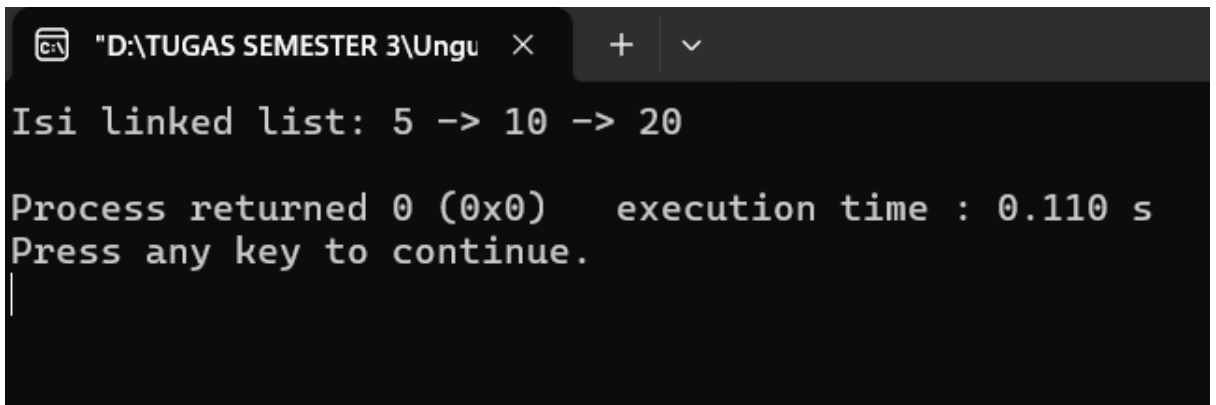
```

```

39     }
40     cout << endl;
41 }
42
43 int main() {
44     Node* head = nullptr;
45
46     insertDepan(head, 10);
47     insertBelakang(head, 20);
48     insertDepan(head, 5);
49
50     cout << "Isi linked list: ";
51     cetakLinkedList(head);
52
53     return 0;
54 }
55

```

Maka Akan Menghasilkan Output



```

"D:\TUGAS SEMESTER 3\Ungu x + v
Isi linked list: 5 -> 10 -> 20
Process returned 0 (0x0) execution time : 0.110 s
Press any key to continue.

```

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

Kode Program:

```
main.cpp x main.cpp x main.cpp x
1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  void insertDepan(Node*& head, int nilai) {
10     Node* newNode = new Node();
11     newNode->data = nilai;
12     newNode->next = head;
13     head = newNode;
14 }
15
16 void insertBelakang(Node*& head, int nilai) {
17     Node* newNode = new Node();
18     newNode->data = nilai;
19     newNode->next = nullptr;
20
21     if (head == nullptr) {
22         head = newNode;
23     } else {
24         Node* temp = head;
25         while (temp->next != nullptr) {
26             temp = temp->next;
27         }
28         temp->next = newNode;
29     }
30 }
31
32 void hapusNode(Node*& head, int nilai) {
33     if (head == nullptr) {
34         cout << "Linked list kosong, tidak ada yang bisa dihapus." << endl;
35         return;
36     }
37
38     if (head->data == nilai) {
39         Node* temp = head;
```

```

main.cpp x main.cpp x main.cpp x
39     Node* temp = head;
40     head = head->next;
41     delete temp;
42     return;
43 }
44
45 Node* temp = head;
46 while (temp->next != nullptr && temp->next->data != nilai) {
47     temp = temp->next;
48 }
49
50 if (temp->next != nullptr) {
51     Node* nodeToDelete = temp->next;
52     temp->next = nodeToDelete->next;
53     delete nodeToDelete;
54 } else {
55     cout << "Node dengan nilai " << nilai << " tidak ditemukan." << endl;
56 }
57 }
58
59 void cetakLinkedList(Node* head) {
60     Node* temp = head;
61     while (temp != nullptr) {
62         cout << temp->data;
63         temp = temp->next;
64         if (temp != nullptr)
65             cout << " -> ";
66     }
67     cout << endl;
68 }
69
70 int main() {
71     Node* head = nullptr;
72
73     insertDepan(head, 10);
74     insertBelakang(head, 20);
75     insertDepan(head, 5);
76
77     cout << "Linked list sebelum penghapusan: ";

```

```

77     cout << "Linked list sebelum penghapusan: ";
78     cetakLinkedList(head);
79
80     hapusNode(head, 10);
81
82     cout << "Linked list setelah penghapusan: ";
83     cetakLinkedList(head);
84
85     return 0;
86 }
87

```

Maka Akan Menghasilkan Output

```
"D:\TUGAS SEMESTER 3\Ungu x + v
Linked list sebelum penghapusan: 5 -> 10 -> 20
Linked list setelah penghapusan: 5 -> 20

Process returned 0 (0x0)    execution time : 0.080 s
Press any key to continue.
|
```

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

Kode Program:

```
main.cpp X main.cpp X main.cpp X main.cpp X
1      #include <iostream>
2      using namespace std;
3
4      struct Node {
5          int data;
6          Node* next;
7      };
8
9      void insertDepan(Node*& head, int nilai) {
10         Node* newNode = new Node();
11         newNode->data = nilai;
12         newNode->next = head;
13         head = newNode;
14     }
15
16     void insertBelakang(Node*& head, int nilai) {
17         Node* newNode = new Node();
18         newNode->data = nilai;
19         newNode->next = nullptr;
20
21         if (head == nullptr) {
22             head = newNode;
23         } else {
24             Node* temp = head;
25             while (temp->next != nullptr) {
26                 temp = temp->next;
27             }
28             temp->next = newNode;
29         }
30     }
31
32     bool cariNode(Node* head, int nilai) {
33         Node* temp = head;
34         while (temp != nullptr) {
35             if (temp->data == nilai) {
36                 return true;
37             }
38             temp = temp->next;
39         }
40         return false;
41     }
42
43     int hitungPanjang(Node* head) {
44         int count = 0;
45         Node* temp = head;
46         while (temp != nullptr) {
47             count++;
48             temp = temp->next;
49         }
50         return count;
51     }
52
53     int main() {
54         Node* head = nullptr;
55
56         insertDepan(head, 10);
57         insertBelakang(head, 20);
58         insertDepan(head, 5);
59
60         int nilaiCari = 20;
61         if (cariNode(head, nilaiCari)) {
62             cout << "Node dengan nilai " << nilaiCari << " ditemukan." << endl;
63         } else {
64             cout << "Node dengan nilai " << nilaiCari << " tidak ditemukan." << endl;
65         }
66
67         int panjang = hitungPanjang(head);
68         cout << "Panjang linked list: " << panjang << endl;
69
70         return 0;
71     }
72 }
```

Maka Akan Menghasilkan Output:

```
"D:\TUGAS SEMESTER 3\Ungu × + v
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

Process returned 0 (0x0)    execution time : 0.090 s
Press any key to continue.
|
```

5. Kesimpulan

1. Single linked list memungkinkan penyimpanan dan pengelolaan data yang dinamis dengan menggunakan pointer.
2. Operasi dasar seperti insert, delete, dan search dapat dilakukan dengan efisien pada single linked list.
3. Penggunaan pointer memudahkan implementasi linked list yang fleksibel dan dinamis.