

LAPORAN PRAKTIKUM
Modul V
“Single Linked List Bagian Kedua”



Disusun Oleh:
Zivana Afra Yulianto - 2211104039
SE-07-02

Dosen:
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

- Memahami dan mengimplementasikan operasi dasar pada struktur data Single Linked List, seperti inialisasi list, menambahkan elemen di awal, menampilkan elemen, serta menghapus elemen.
- Mempelajari operasi tambahan seperti mencari elemen dengan nilai tertentu, menghitung jumlah semua elemen dalam list, serta menemukan elemen maksimum dan minimum.
- Mengembangkan kemampuan dalam alokasi dan dealokasi memori dinamis menggunakan pointer di C++.

2. Landasan Teori

- Single Linked List

Single Linked List adalah salah satu jenis struktur data linear yang terdiri dari serangkaian elemen yang disebut node. Setiap node menyimpan data serta alamat dari node berikutnya dalam daftar. Operasi umum yang dilakukan pada single linked list meliputi penambahan elemen, penghapusan, pencarian, serta traversal.

- Node dan Pointer

Pada single linked list, setiap node terdiri dari dua bagian utama:

- Info: Menyimpan data dari node tersebut.
- Next: Pointer yang menunjuk ke node berikutnya. Pointer ini memungkinkan elemen-elemen list dihubungkan satu sama lain, sehingga membentuk suatu urutan.

- Alokasi dan Dealokasi Memori

- Alokasi Memori adalah proses pemesanan memori untuk menyimpan node baru menggunakan operator new di C++.
- Dealokasi Memori dilakukan untuk melepaskan memori yang digunakan oleh suatu node yang tidak lagi diperlukan dengan menggunakan delete.

- Operasi Dasar Single Linked List

- a. Insert First: Menambahkan node baru di awal list, di mana node baru akan menjadi node pertama, dan node sebelumnya menjadi node kedua.
- b. Find Element: Proses pencarian node dalam linked list berdasarkan nilai (info) yang disimpan di dalam node tersebut.
- c. Sum Info: Menghitung jumlah total semua nilai yang ada pada setiap node dalam linked list.
- d. Find Max/Min: Mencari node dengan nilai maksimum atau minimum dalam linked list.

- Keuntungan Linked List

Linked list memiliki keuntungan dalam hal fleksibilitas ukuran. Berbeda dengan array yang ukurannya tetap, linked list dapat bertambah atau berkurang secara dinamis. Namun, linked list tidak mendukung akses elemen secara langsung, sehingga traversal harus dilakukan dari awal list.

3. Guided

Code :

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

// Struktur untuk node dalam 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;

    if (tail == nullptr)
    {
        tail = new_node;
    }
}

// Fungsi untuk menambahkan elemen baru 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 = head;
    head = new_node;

    if (tail == nullptr)
    {
        tail = new_node;
    }
}

// Fungsi untuk mencari elemen dalam linked list
int findElement(Node *head, int x)
{
    Node *current = head;
    int index = 0;

    while (current != nullptr)
    {
        if (current->data == x)
        {
            return index;
        }
        current = current->next;
        index++;
    }
    return -1;
}

// Fungsi untuk menampilkan elemen dalam linked list
void display(Node *node)
{
    while (node != nullptr)
    {
        cout << node->data << " ";
        node = node->next;
    }
    cout << endl;
}

// Fungsi untuk menghapus elemen dari linked list
void deleteElement(Node *&head, int x)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong" << endl;
        return;
    }
}
```

```

    if (head->data == x)
    {
        Node *temp = head;
        head = head->next;
        delete temp;
        return;
    }
    Node *current = head;
    while (current->next != nullptr)
    {
        if (current->next->data == x)
        {
            Node *temp = current->next;
            current->next = current->next->next;
            delete temp;
            return;
        }
        current = current->next;
    }
}

int main()
{
    Node *head = nullptr;
    Node *tail = nullptr;

    insertFirst(head, tail, 3);
    insertFirst(head, tail, 5);
    insertFirst(head, tail, 7);

    insertLast(head, tail, 11);
    insertLast(head, tail, 14);
    insertLast(head, tail, 15);

    cout << "Elemen dalam linked list: ";
    display(head);

    int x;
    cout << "Masukkan elemen yang ingin dicari ";
    cin >> x;

    int result = findElement(head, x);

    if (result == -1)
        cout << "Elemen tidak ditemukan dalam linked list" << endl;
    else
        cout << "Elemen ditemukan pada indeks" << result << endl;

    cout << "Masukkan elemen yang ingin dihapus: ";
    cin >> x;
    deleteElement(head, x);

    cout << "Elemen dalam linked list setelah penghapusan: ";
    display(head);
    return 0;
}

```

Screenshot Output :

```

Elemen dalam linked list: 15 14 11 7 5 3
Masukkan elemen yang ingin dicari 15
Elemen ditemukan pada indeks0
Masukkan elemen yang ingin dihapus: 14
Elemen dalam linked list setelah penghapusan: 15 14 7 5 3
PS C:\STD_Zivana_Afra_Yulianto>

```

Deskripsi :

Kode ini mengimplementasikan Linked List dasar dengan beberapa fungsi untuk menambahkan, mencari, menampilkan, dan menghapus elemen di dalamnya. Berikut adalah deskripsi dari tiap bagian kode:

1. Struktur Node:
 - Node adalah struktur yang merepresentasikan satu elemen dalam linked list. Setiap node memiliki dua bagian:
 - data: menyimpan data berupa integer.
 - next: pointer yang menunjuk ke node berikutnya.
2. Fungsi `insertFirst`:
 - Fungsi ini digunakan untuk menambahkan elemen baru (`new_data`) ke awal dari linked list.
 - Jika linked list masih kosong (tail adalah `nullptr`), elemen baru juga akan menjadi tail.
3. Fungsi `insertLast`:
 - Fungsi ini menambahkan elemen baru (`new_data`) ke akhir linked list. Namun, implementasinya tampaknya memiliki kesalahan karena `new_node->next` diatur ke `head`, yang salah karena seharusnya `new_node` ditambahkan di akhir daftar, bukan menghubungkannya kembali ke `head`.
4. Fungsi `findElement`:
 - Fungsi ini mencari elemen dengan nilai tertentu (`x`) dalam linked list. Fungsi akan mengembalikan indeks dari elemen yang dicari atau -1 jika elemen tidak ditemukan.
5. Fungsi `display`:
 - Fungsi ini menampilkan semua elemen dalam linked list mulai dari node pertama hingga node terakhir.
6. Fungsi `deleteElement`:
 - Fungsi ini bertujuan untuk menghapus elemen dengan nilai tertentu (`x`) dari linked list. Namun, implementasinya belum lengkap dan baru menangani kondisi di mana linked list kosong.

4. Unguided

singlelist.h :

```
#ifndef SINGLELIST_H
#define SINGLELIST_H
#include <iostream>

typedef int infotype;
typedef struct ElmList *address;

struct ElmList
{
    infotype info;
    address next;
};

struct List
{
    address First;
};

// Basic operations
void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertFirst(List &L, address P);

// Additional operations for the exercises
address findElm(List L, infotype x);
int sumInfo(List L);
address findMax(List L);
address findMin(List L);

#endif
```

singlelist.cpp :

```
#include "singlelist.h"
#include <iostream>
using namespace std;

void CreateList(List &L)
{
    L.First = NULL;
}

address alokasi(infotype x)
{
    address P = new ElmList;
    P->info = x;
    P->next = NULL;
    return P;
}

void dealokasi(address &P)
{
    delete P;
    P = NULL;
}

void printInfo(List L)
{
    address P = L.First;
    cout << "Isi List: ";
    while (P != NULL)
    {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

void insertFirst(List &L, address P)
{
    P->next = L.First;
    L.First = P;
}
```

```

// Exercise 3: Find element with specific info
address findElm(List L, infotype x)
{
    address P = L.First;
    while (P != NULL)
    {
        if (P->info == x)
        {
            return P;
        }
        P = P->next;
    }
    return NULL;
}

// Exercise 4: Calculate sum of all info
int sumInfo(List L)
{
    address P = L.First;
    int sum = 0;
    while (P != NULL)
    {
        sum += P->info;
        P = P->next;
    }
    return sum;
}

// Additional helper functions
address findMax(List L)
{
    if (L.First == NULL)
        return NULL;

    address P = L.First;
    address maxAddr = P;

    while (P != NULL)
    {
        if (P->info > maxAddr->info)
        {
            maxAddr = P;
        }
        P = P->next;
    }
    return maxAddr;
}

address findMin(List L)
{
    if (L.First == NULL)
        return NULL;

    address P = L.First;
    address minAddr = P;

    while (P != NULL)
    {
        if (P->info < minAddr->info)
        {
            minAddr = P;
        }
        P = P->next;
    }
    return minAddr;
}

```

main.cpp :

```
// main.cpp
#include "singlelist.h"
#include <iostream>
using namespace std;

int main() {
    List L;
    address P;

    // Create list and insert elements as per exercise requirements
    CreateList(L);

    // Insert elements: 9, 12, 8, 0, 2
    P = alokasi(2);
    insertFirst(L, P);
    P = alokasi(0);
    insertFirst(L, P);
    P = alokasi(8);
    insertFirst(L, P);
    P = alokasi(12);
    insertFirst(L, P);
    P = alokasi(9);
    insertFirst(L, P);

    // Exercise 2: Display all elements
    cout << "\nIsi List : ";
    printInfo(L);

    // Exercise 3: Find element with info = 8
    cout << "\nMencari angka 8" << endl;
    address found = findElm(L, 8);
    if (found != NULL) {
        cout << "Element 8 found in the list" << endl;
    } else {
        cout << "Element 8 not found in the list" << endl;
    }

    // Exercise 4: Calculate and display sum of all info
    cout << "\nMenghitung total info seluruh elemen" << endl;
    int total = sumInfo(L);
    cout << "Total sum of all elements: " << total << endl;

    // Additional information
    address maxElement = findMax(L);
    address minElement = findMin(L);

    if (maxElement != NULL) {
        cout << "\nMaximum element: " << maxElement->info << endl;
    }
    if (minElement != NULL) {
        cout << "Minimum element: " << minElement->info << endl;
    }

    // Clean up
    while (L.First != NULL) {
        P = L.First;
        L.First = L.First->next;
        dealokasi(P);
    }

    return 0;
}
```


Shreenshoot Output :

```
Isi List: 9 12 8 0 2

Mencari angka 8
Elemen 8 ditemukan didalam list

Menghitung total info seluruh elemen
TJumlah total semua elemen adalah : 31
PS C:\STD_Zivana_Afra_Yulianto>
```

Deskripsi :

Header File (singlelist.h):

- Struktur Data:
 - ElmList: Struktur untuk node yang menyimpan data (info) dan pointer (next) ke node berikutnya.
 - List: Struktur yang memiliki pointer ke node pertama (First).
- Operasi Dasar:
 - CreateList(List &L): Menginisialisasi list kosong.
 - alokasi(info type x): Mengalokasikan memori untuk node baru dengan data x.
 - dealokasi(address &P): Menghapus node dari memori.
 - printInfo(List L): Menampilkan semua elemen dalam list.
 - insertFirst(List &L, address P): Menambahkan node baru di awal list.
- Operasi Tambahan:
 - findElm(List L, info type x): Mencari node dengan nilai x.
 - sumInfo(List L): Menghitung jumlah semua elemen dalam list.

2. Implementation File (singlelist.cpp):

- Fungsi CreateList(List &L): Menginisialisasi list kosong dengan mengatur L.First = NULL.
- Fungsi alokasi(info type x): Mengalokasikan memori untuk node baru dengan nilai x dan mengembalikan alamatnya.
- Fungsi dealokasi(address &P): Menghapus node dari memori dan mengatur pointer ke NULL.
- Fungsi printInfo(List L): Menampilkan semua elemen dalam list.
- Fungsi insertFirst(List &L, address P): Menambahkan node baru di awal list dengan menghubungkan node baru ke node yang sebelumnya berada di awal.

- Fungsi `findElm(List L, infotype x)`: Mencari node dengan nilai tertentu. Jika ditemukan, mengembalikan alamatnya, jika tidak, mengembalikan `NULL`.
- Fungsi `sumInfo(List L)`: Menghitung jumlah semua elemen dalam list dan mengembalikan hasilnya.
- Fungsi `findMax(List L)` dan `findMin(List L)`: Menemukan node dengan nilai maksimum dan minimum dalam list.

3. Main File (`main.cpp`):

- Fungsi `main()`:
 - Membuat list baru dan menambahkan elemen-elemen (9, 12, 8, 0, 2) ke dalam list.
 - Menampilkan isi list menggunakan `printInfo()`.
 - Mencari elemen dengan nilai 8 menggunakan `findElm()` dan menampilkan hasil pencarian.
 - Menghitung dan menampilkan total jumlah elemen dengan `sumInfo()`.
 - Setelah semua operasi selesai, list dibersihkan dengan menghapus semua node menggunakan `dealokasi()`.

Diatas merupakan keseluruhan dari code jika digabungkan. Berikut merupakan code yang sudah dibagi per nomor soal.

1. Menampilkan isi list

Code:

```
// NOMOR 1
cout << "\nIsi List : ";
printInfo(L);
```

Screenshoot :

```
Isi List: 9 12 8 0 2
PS C:\STD_Zivana_Afra_Yulianto>
```

Deskripsi :

Code ini merupakan code yang cukup sederhana dikarenakan hanya menampilkan isi dari List menggunakan perintah `printInfo(L)`

2. Mencari elemen angka 8

Code :

```
// NOMOR 2
cout << "\nMencari angka 8" << endl;
address found = findElm(L, 8);
if (found != NULL)
{
    cout << "Elemen 8 ditemukan didalam list" << endl;
}
else
{
    cout << "Elemen 8 tidak ditemukan dalam list" << endl;
}
```

Screenshoot :

```
Mencari angka 8
Elemen 8 ditemukan didalam list
PS C:\STD_Zivana_Afra_Yulianto>
```

Deskripsi :

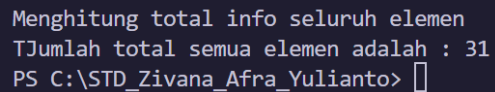
Code ini berfungsi untuk mencari elemen pada List, pada kasus ini, dicari angka 8

3. Menjumlahkan seluruh element yang ada dalam list

Code :

```
// NOMOR 3
cout << "\nMenghitung total info seluruh elemen" << endl;
int total = sumInfo(L);
cout << "TJumlah total semua elemen adalah : " << total << endl;
```

Screenshoot :



```
Menghitung total info seluruh elemen
TJumlah total semua elemen adalah : 31
PS C:\STD_Zivana_Afra_Yulianto>
```

Deskripsi :

Code ini berfungsi untuk menjumlahkan semua elemen yang ada menggunakan `sumInfo(L)` kemudian menampilkannya.

5. Kesimpulan

- Operasi Dasar Single Linked List berhasil diimplementasikan dengan baik. Fungsi seperti `CreateList`, `insertFirst`, dan `printInfo` dapat digunakan untuk membuat list, menambah elemen di awal, serta menampilkan isi dari list tersebut.
- Operasi Tambahan seperti pencarian elemen (`findElm`), penjumlahan nilai elemen (`sumInfo`), serta pencarian elemen maksimum dan minimum (`findMax` dan `findMin`) juga berhasil dilakukan. Hasil yang diperoleh sesuai dengan nilai yang diinputkan dalam linked list.
- Alokasi dan dealokasi memori dinamis menggunakan pointer pada C++ menjadi komponen penting dalam pengelolaan linked list. Proses dealokasi memastikan tidak ada kebocoran memori setelah node-node dalam list tidak lagi dibutuhkan.
- Praktikum ini juga mengajarkan pentingnya memahami cara kerja pointer dalam linked list, karena pointer inilah yang menghubungkan node-node dan memungkinkan operasi-operasi dilakukan secara dinamis tanpa batasan ukuran memori yang tetap.