LAPORAN PRAKTIKUM

MODUL 4

“SINGLE LINKED LIST (BAGIAN KEDUA)”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

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.

2. Landasan teori

Pencarian dalam single linked list (SLL) biasanya dilakukan dengan menelusuri setiap node mulai dari awal hingga menemukan data yang dicari atau mencapai akhir dari list. Karena single linked list hanya memiliki referensi ke node berikutnya (tidak ada referensi ke node sebelumnya), pencarian bersifat linier, yaitu O(n), dengan \(n \) adalah jumlah node dalam list. Dengan melakukan operasi searching, operasi-operasi seperti insert after, delete after, dan update akan lebih mudah.

Berikut pseudocode dari searching:

```
function search(head, target):
  current = head
  while current is not None:
    if current.data == target:
      return True  // Data ditemukan
    current = current.next
  return False  // Data tidak ditemukan
```

Penjelasan:
Head adalah node pertama dari linked list.
Current adalah variabel yang digunakan untuk melintasi setiap node.
Target adalah nilai yang dicari.
Jika data ditemukan, fungsi akan mengembalikan `True’; jika tidak, fungsi akan mengembalikan `False` setelah mencapai akhir list.

3. Guided

Program ini merupakan implementasi operasi dasar pada single linked list yang menggunakan penyisipan, pencarian, penghapusan, dan penelusuran elemen dalam daftar. Pertama-tama dibuat srtruktur node yang digunakan untuk merepresentasikan elemen atau node dari linked list. Setiap Node memiliki dua bagian yaitu, data sebagai penyimpan nilai dari elemen. Lalu, next yang merupakan pointer yang menunjuk ke node berikutnya dalam linked list. Kemudian digunakan fungsi insert first untuk menambahkan elemen baru di awal linked list. Dengan membuat node baru yang menyimpan data baru, kemudian

dihubungkan node baru ke node pertama (head) sebelumnya dan memperbarui head menjadi node baru. Jika linked list kosong, tail diperbarui menjadi node baru.

```
// Struktur untuk node dalam linked list
struct Node {
    int data;
    Node* next;
};

// fungsi untuk menambahkan elemen baru ke awal linkded 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;
    head = new_node;

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

Selanjutnya digunakan fungsi insert last yang menambahkan elemen baru diakhir linked list. Jika linked list kosong maka node baru menjadi head dan tail, dan jika tidak kosong fungsi menambahkan node baru setelah tail yang ada dan memperbarui tai menjadi node baru.

```
// fungsi 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 = nullptr;

    if (head == nullptr){
        head = new_node;
        tail = new_node;
    } else {
        tail->next = new_node;
        tail = new_node;
    }
}
```

Digunakan juga fungsi searching sebuah elemen dalam linked list. Untuk memulai pencarian dari head dan mengambil indeks elemen jika ditemukan, sedangakan jika elemen tidak ditemukan fungsi mengembalikan -1.

```
// fungsi 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;
}
```

Dibuat juga fungsi untuk mencetak semua elemen dalam linked list mulai dari head hingga akhir. Digunakan cout untuk menampilkan data dari setiap node.

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

Salah satu fungsi yang digunakan juga adalah delete element untuk menghapus node yang berisi sebuah nilai. Dimana jika head adalah elemen yang dicari maka head akan dipindah ke node berikutnya dan node sebelumnya dihapus, jika nilai tersebut berada di node selain head maka fungsi menemuka node tersebut dan memperbarui pointer untuk melewati node tersebut lalu menghapus node yang memiliki nilai tersebut.

```
// fungsi 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;
    }
}
```

Pada fungsi utama untuk menjalankan programnya, dibuat linked list dengan menambahkan beberapa elemen menggunakan insert first dan insert last. Dibuat fungsi menampilkan elemen dalam linked list. Kemudian meminta pengguna untuk mencari elemen, kemudian menampilkan posisi elemen tersebut jika ditemukan. Meminta pengguna untuk menghapus elemen tertentu, lalu menampilkan kembali linked list setelah penghapusan.

```cpp
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, 18);

    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;
}
```

4. Unguided

1. Pada program membuat implementasi ADT Single Linked List, dibuat sebuah file header `singlelist.h` yang berisi definisi tipe data, struktur untuk elemen list dan list, dan deklarasi semua prosedur dan fungsi. File header ini berfungsi sebagai kontrak atau blueprint yang mendefinisikan apa saja yang ada dalam implementasi single linked list, tanpa menuliskan detail implementasinya. Ini membuat kode lebih terorganisir dan mudah dikelola.

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmList* address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

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

#endif
```

Setelah itu dibuat lagi sebuah file source code yang dinamakan singlelist.cpp sebagai tempat untuk menjalankan program, yang berisikan, memanggil semua fungsi dan prosedur (createlist, alokasi, dealokasi, printinfo, dan insert first) dan program utama untuk menguji implementasi.

```cpp
#include <iostream>
#include "singlelist.h"
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;
    while (P != NULL) {
        cout << P->info;
        if (P->next != NULL) {
            cout << " ";
        }
        P = P->next;
    }
    cout << endl;
}

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

// Program utama untuk menguji implementasi
int main() {
    List L;
    CreateList(L);

    // Membuat linked list sesuai gambar (9->12->8->0->2)
    address P1 = alokasi(2);
    insertFirst(L, P1);

    address P2 = alokasi(0);
    insertFirst(L, P2);

    address P3 = alokasi(8);
    insertFirst(L, P3);

    address P4 = alokasi(12);
    insertFirst(L, P4);

    address P5 = alokasi(9);
    insertFirst(L, P5);

    cout << "";
    printInfo(L);

    return 0;
}
```

Program akan membuat linked list sesuai dengan gambar dan menampilkan output berikut:

```
PS D:\tugas
9 12 8 0 2
PS D:\tugas
```

2. Untuk fungsi mencari sebuah elemen tambahkan beberapa fungsi terlebih dahulu. Pada file singlelist.h pada bagian deklarasi fungsinya dibuat seperti berikut:

```
address findElm(List L, infotype x); //2 deklarasi fungsi mencari elemen 8
```

Lalu, pada file singlelist.cpp, dibuat implementasi fungsi mencari elemen tersebut. Pada fungsi tersebut dapat menerima parameter List L dan infotype x (nilai yang dicari), lalu mengembalikan address (pointer ke node) yang berisi nilai yang dicari, jika tidak maka mengembalikan NULL jika nilai tidak ditemukan.

```
//2 implementasi fungsi mencari elemen
address findElm(List L, infotype x){
    address P = L.First;
    while (P != NULL && P->info != x){
        P = P->next;
    }
    return P;
}
```

Pada program utama yang mendemonstrasikan pembuatan linked list seperti sebelumnya tetapi tambahkan pencarian elemen dengan info 8 dengan dibuat statement jika yang dicari tidak sama dengan NULL maka nilai x ditemukan dalam list, sebaliknya jika tidak maka nilai x tidak ditemuakan dalam list seperti berikut

```
// 2 mencari elemen dengan info 8
infotype x = 8;
address found = findElm(L, x);
if (found != NULL){
    cout << x << " ditemukan dalam list" << endl;
} else {
    cout << x << " tidak ditemukan dalam list" << endl;
}
```

Output dari program akan menjadi seperti berikut:

```
8 ditemukan dalam list
```

3. Sama dengan nomor 2 untuk dapat menghitung total info seluruh elemen tambahkan fungsi untuk menghitung info dari semua elemen pada linked list pada file header bagian deklarasi fungsi `int sumInfo(List L);`

   Lalu implementasikan fungsi sumInfo di file singlelist.cpp, yang dapat menerima parameter List L, menghitung total nilai info dari semua elemen, dan mengembalikan nilai total tersebut

```
//3 implementasi fungsi menjumlahkan seluruh info elemen
int sumInfo(List L){
    int total = 0;
    address P = L.First;

    while (P != NULL){
        total += P->info;
        P = P->next;
    }
    return total;
}
```

Program utama yang mendemonstrasikan pembuatan linked list tadi dan tambahkan perhitungan total info.

```
// 3 menghitung total info
int total = sumInfo(L);
cout << "Total info dari kelima elemen adalah " << total << endl;
```

Output program akan menampilkan isi linked list dan total dari seluruh info yang ada.

```
Total info dari kelima elemen adalah 31
```

5. Kesimpulan

Dari praktikum ini, dapat disimpulkan bahwa single linked list merupakan struktur data yang sangat berguna untuk mengelola data yang ukurannya dinamis. Operasi-operasi yang telah diimplementasikan, seperti penambahan, penghapusan, dan pencarian, menunjukkan fleksibilitas single linked list. Penggunaan pointer dalam mengelola hubungan antar node serta penerapan konsep ADT melalui pemisahan file header dan source code telah meningkatkan modularitas dan kemudahan pemeliharaan program.