LAPORAN PRAKTIKUM
Modul 5
“Single Lingked List(Bagian kedua)”

Disusun Oleh:
Muhammad Daniel Anugrah Pratama -2311104063
SE07-02

Dosen :
Wahyu Andi Saputra,S.Pd, M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

5. Memahami penggunaan linked list dengan pointer operator- operator dalam program.
6. Memahami operasi-operasi dasar dalam linked list.
7. Membuat program dengan menggunakan linked list dengan prototype yang ada

2. Landasan Teori

A.Searching

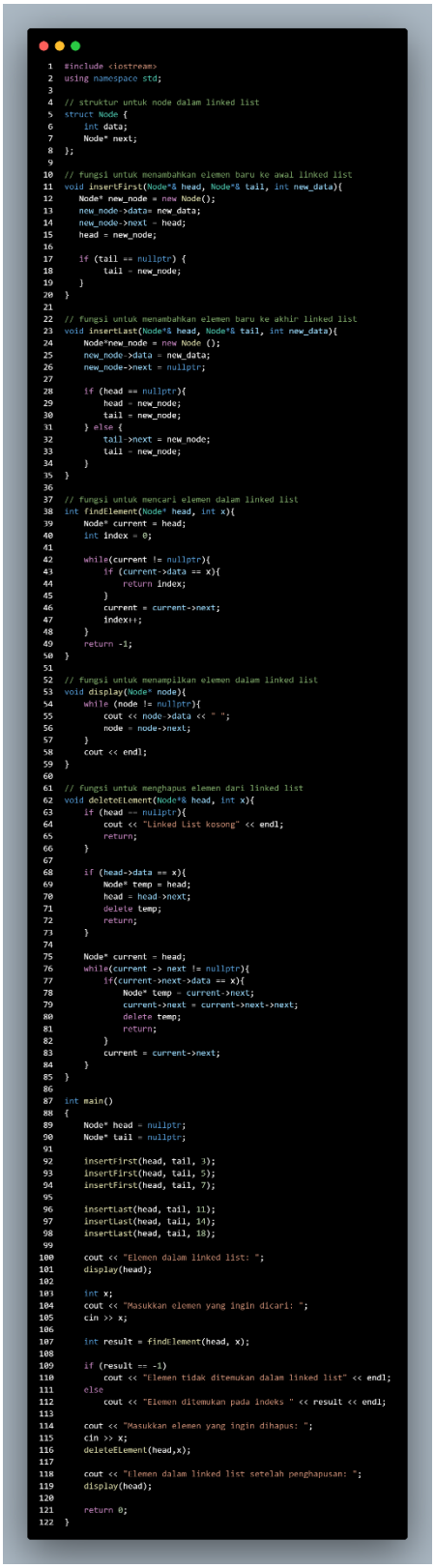
Searching merupakan operasi dasar pada linked list yang dilakukan untuk mencari node tertentu. Proses ini berjalan dengan mengunjungi setiap node satu per satu hingga node yang dicari ditemukan. Dengan melakukan operasi searching, operasi-operasi lainnya seperti insert after, delete after, dan update akan lebih mudah dilaksanakan.

Proses searching dimulai dari node pertama (head) dan terus melanjutkan ke node berikutnya (next) hingga mencapai node yang dicari atau hingga mencapai akhir linked list (nullptr). Jika node yang dicari ditemukan, operasi dapat dilanjutkan sesuai kebutuhan; jika tidak, maka hasilpencarian menunjukkan bahwa node tersebut tidak ada dalam linked list.

Semua fungsi dasar yang terkait dengan operasi searching, serta operasi lainnya, merupakan bagian dari Abstract Data Type (ADT) dari single linked list. Dalam implementasi menggunakan bahasa pemrograman C++, semua ADT tersebut biasanya disimpan dalam file dengan ekstensi .c untuk kode sumber dan .h untuk header file.

3. Guided

Kode Program:



```cpp
#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 = nullptr;

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

Outputnya:

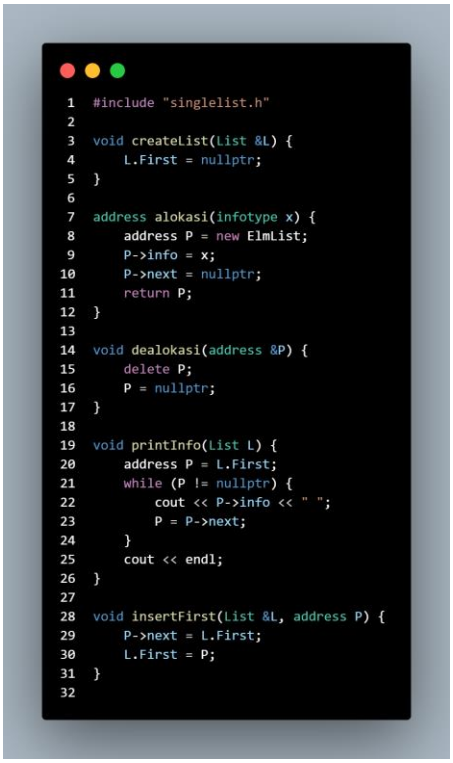
```
PS C:\Users\usER\OneDrive\Documents\per6sd> cd "c:\Users\usER\OneDrive\
5 } ; if ($?) { .\Guidedm5 }
Elemen dalam linked list: 7 5 3 11 14 18
Masukkan elemen yang ingin dicari: 5
Elemen ditemukan pada indeks 1
Masukkan elemen yang ingin dihapus: 11
Elemen dalam linked list setelah penghapusan: 7 5 3 14 18
PS C:\Users\usER\OneDrive\Documents\per6sd\.vscode>
```

4. Unguided

1. Buatlah ADT Single Linked list sebagai berikut di dalam file “singlelist.h”

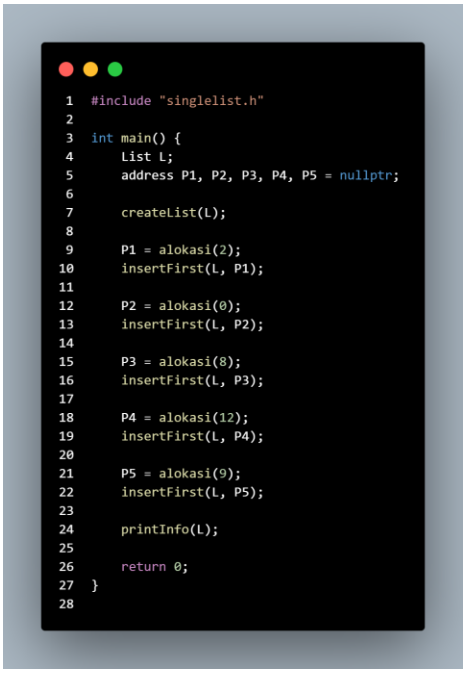
```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

#include <iostream>
using namespace std;

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Deklarasi fungsi dan prosedur
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertFirst(List &L, address P);

#endif
```

buat implementasi ADT Single Linked list pada file “singlelist.cpp”.
Adapun isi data



```cpp
#include "singlelist.h"

void createList(List &L) {
    L.First = nullptr;
}

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

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

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

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

```

h hasil implementasi ADT pada file “main.cpp”



```cpp
#include "singlelist.h"

int main() {
    List L;
    address P1, P2, P3, P4, P5 = nullptr;

    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;
}

```

Outputnya:

```
9 12 8 0 2
```

2. Carilah elemen dengan info 8 dengan membuat fungsi baru.
   fungsi findElm(L : List, x : infotype) : address
   Kode Program:

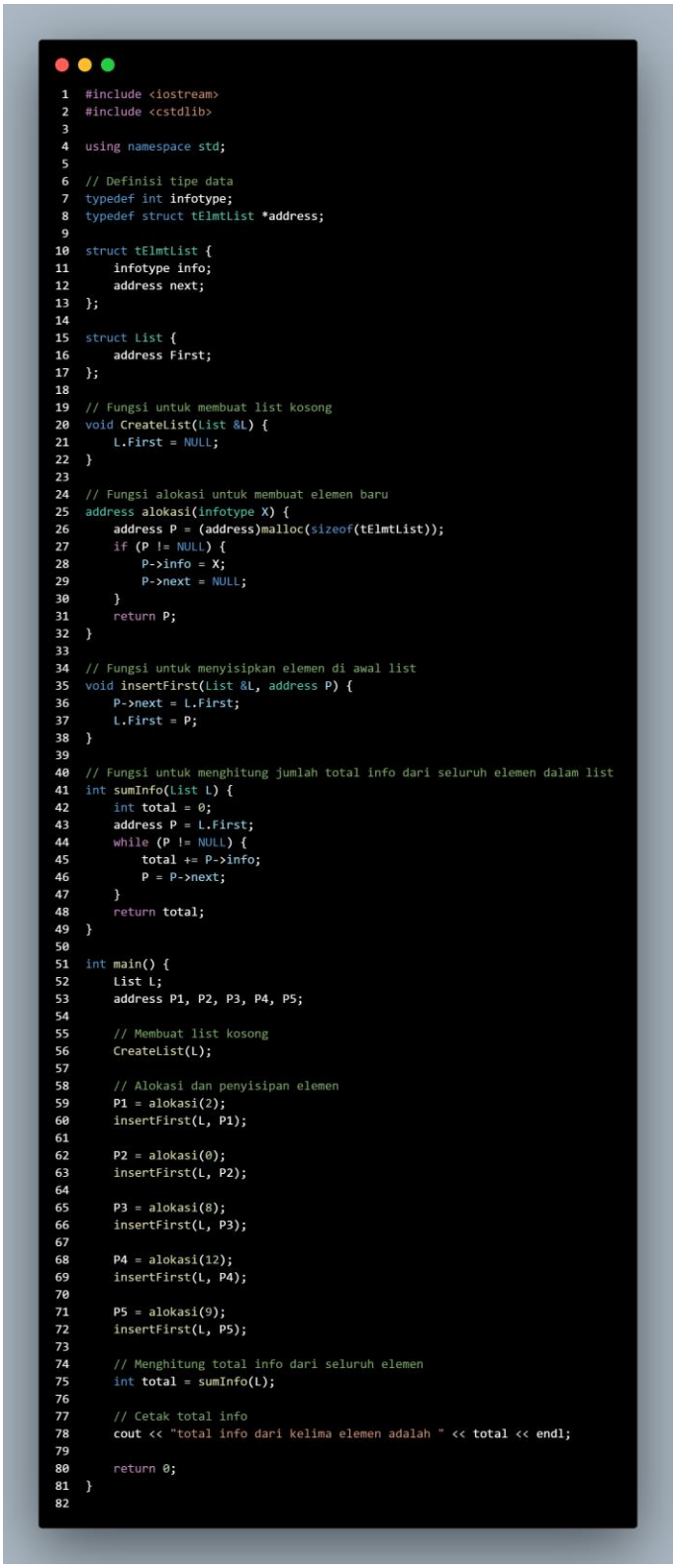
```cpp
#include <iostream>
#include <vector>

using namespace std;

// Asumsikan infotype adalah integer
int findElm(const vector<int>& L, int x) {
    // Menggunakan algoritma pencarian linear
    for (int i = 0; i < L.size(); ++i) {
        if (L[i] == x) {
            return i; // Mengembalikan indeks jika ditemukan
        }
    }
    return -1; // Mengembalikan -1 jika tidak ditemukan
}

int main() {
    vector<int> myList = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6, 2, 6, 4, 3, 3, 8, 3, 2, 7, 9, 5};
    int target = 8;

    int result = findElm(myList, target);

    if (result != -1) {
        cout << target << " ditemukan dalam list" << endl;
    } else {
        cout << target << " tidak ditemukan dalam list" << endl;
    }

    return 0;
}
```

Outputnya:

```
PS C:\Users\usER\OneDrive\Documents\per6sd> cd "c:\
ement_list.cpp -o find_element_list } ; if ($?) { .
8 ditemukan dalam list
PS C:\Users\usER\OneDrive\Documents\per6sd\.vscode>
```

3. Hitunglah jumlah total info seluruh elemen (9+12+8+0+2=31).
   Kode program:

Telkom University Purwokerto



```cpp
#include <iostream>
#include <cstdlib>

using namespace std;

// Definisi tipe data
typedef int infotype;
typedef struct tElmtList *address;

struct tElmtList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Fungsi untuk membuat list kosong
void CreateList(List &L) {
    L.First = NULL;
}

// Fungsi alokasi untuk membuat elemen baru
address alokasi(infotype X) {
    address P = (address)malloc(sizeof(tElmtList));
    if (P != NULL) {
        P->info = X;
        P->next = NULL;
    }
    return P;
}

// Fungsi untuk menyisipkan elemen di awal list
void insertFirst(List &L, address P) {
    P->next = L.First;
    L.First = P;
}

// Fungsi untuk menghitung jumlah total info dari seluruh elemen dalam list
int sumInfo(List L) {
    int total = 0;
    address P = L.First;
    while (P != NULL) {
        total += P->info;
        P = P->next;
    }
    return total;
}

int main() {
    List L;
    address P1, P2, P3, P4, P5;

    // Membuat list kosong
    CreateList(L);

    // Alokasi dan penyisipan elemen
    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);

    // Menghitung total info dari seluruh elemen
    int total = sumInfo(L);

    // Cetak total info
    cout << "total info dari kelima elemen adalah " << total << endl;

    return 0;
}

```

Outputnya:

```
PS C:\Users\usER\OneDrive\Documents\per6sd\.vscode>
 sum_elements.cpp -o sum_elements } ; if ($?) { .\su
total info dari kelima elemen adalah 31
PS C:\Users\usER\OneDrive\Documents\per6sd\.vscode>
```

5. **Kesimpulan**

   Dalam pemrograman, penggunaan linked list sebagai struktur data dinyatakan sangat efektif untuk menangani operasi-operasi dasar seperti penambahan, pencarian, dan penghapusan elemen. Melalui implementasi ADT Single Linked List, kita dapat memahami bagaimana setiap node terhubung dengan menggunakan pointer, serta bagaimana operasi seperti searching memudahkan manipulasi data. Dengan menciptakan berbagai fungsi seperti insertFirst, insertLast, dan deleteElement, kita dapat mengelola data dengan lebih fleksibel. Selain itu, penambahan fungsi untuk mencari elemen tertentu dan menghitung jumlah total dari semua elemen menunjukkan kemampuan linked list dalam melakukan operasi yang lebih kompleks. Secara keseluruhan, penguasaan konsep dan implementasi linked list sangat penting dalam pengembangan algoritma dan aplikasi berbasis data, memberikan fondasi yang kuat untuk pemrograman yang lebih lanjut.