LAPORAN PRAKTIKUM
Modul V
“Single Linked List (Bagian Kedua)”

Disusun Oleh:
Berlian Seva Astryana -2311104067
Kelas
S1SE-07-02

Dosen :
Arief Rais Bahtiar, S.Kom., M.Kom

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. **Tujuan**
   - **1.1** Memahami penggunaan linked list dengan pointer operator- operator dalam program.
   - **1.2** Memahami operasi-operasi dasar dalam linked list.
   - **1.3** Membuat program dengan menggunakan linked list dengan prototype yang ada
2. **Landasan Teori**

   Pada implementasi struktur data seperti single linked list, terdapat operasi dasar yang mencakup searching, insertion, deletion, dan update. Dalam linked list, operasi pencarian atau searching dilakukan dengan menelusuri setiap node dalam list satu per satu hingga node yang diinginkan ditemukan. Proses pencarian ini berhenti ketika node yang dicari ditemukan atau setelah mencapai akhir list jika node tersebut tidak ada. Proses searching merupakan komponen mendasar yang mendukung operasi lain, seperti menambah elemen setelah node tertentu (insert after), menghapus node setelah node tertentu (delete after), dan memperbarui isi dari suatu node (update).

   Operasi-operasi dasar ini membentuk bagian penting dari abstraksi single linked list atau Abstract Data Type (ADT), yang berfungsi menyembunyikan detail implementasi dari pengguna. Dalam implementasinya menggunakan bahasa C, ADT ini biasanya dibagi menjadi dua jenis file, yaitu file dengan ekstensi .c dan .h. File .c berisi definisi fungsi yang menjalankan operasi-operasi tersebut, sementara file .h berisi deklarasi fungsi, tipe data, serta antarmuka yang memungkinkan pemrogram lain untuk menggunakan ADT single linked list tanpa perlu memahami detail pengkodean di baliknya. Hal ini mempermudah pengelolaan dan penggunaan kode, serta memungkinkan pemrograman yang lebih modular, di mana setiap fungsi dasar beroperasi pada satu unit data dalam list, mengoptimalkan efisiensi dan meminimalkan redundansi dalam kode.
3. **Guided**

   Program:

Telkom University Purwokerto

guided

```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 umtuk 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 menghapuselemen dari Linked liast*/
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->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);

    insertFirst(head, tail, 11);
    insertFirst(head, tail, 14);
    insertFirst(head, tail, 18);

    cout << "Elemen dalam linked liat: ";
    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 << "Masukan elemen yang ingin dihapus: ";
    cin >> x;
    deleteElement(head, x);

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

    return 0;
}
```

Output:

```
Elemen dalam linked liat: 18 14 11 7 5 3
Masukkan elemen yang ingin dicari: 11
Elemen ditemukan pada indeks2
Masukan elemen yang ingin dihapus: 3
Elemen dalam linked list setelah penghapusan: 18 14 11 7 5 3
```

4. Unguided

4.1 Program:

singlelist.h

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Prosedur dan fungsi ADT Single Linked List
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertFirst(List &L, address P);

#endif
```

Telkom University Purwokerto

singlelist.cpp

```cpp
#include <iostream>
#include "singlelist.h"

using namespace std;

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

// Fungsi untuk mengalokasi node baru dengan nilai x
address alokasi(infotype x) {
    address P = new ElmList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

// Prosedur untuk mendealokasi node
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Prosedur untuk mencetak info dari list
void printInfo(const List &L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

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

Telkom University Purwokerto

main.cpp

```cpp
#include <iostream>
#include "singlelist.h"

using namespace std;

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

    createList(L);

    // Alokasi dan memasukkan nilai sesuai urutan
dalam ilustrasi
    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);

    // Mencetak informasi dari List
    printInfo(L);

    return 0;
}
```

Output:

```
9 12 8 0 2
```

1.1 Program:

singlelist.h

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Prosedur dan fungsi ADT Single Linked List
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertFirst(List &L, address P);
address findElm(const List &L, infotype x); //
Deklarasi fungsi findElm

#endif
```

Telkom University Purwokerto

singlelist.cpp

```cpp
#include <iostream>
#include "singglelist.h"

using namespace std;

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

// Fungsi untuk mengalokasi node baru dengan nilai
x
address alokasi(infotype x) {
    address P = new ElmList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

// Prosedur untuk mendealokasi node
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Prosedur untuk mencetak info dari list
void printInfo(const List &L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

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

// Fungsi untuk mencari elemen dengan nilai x
dalam list
address findElm(const List &L, infotype x) {
    address P = L.First;
    while (P != NULL) {
        if (P->info == x) {
            return P; // Mengembalikan alamat node
jika ditemukan
        }
        P = P->next;
    }
    return NULL; // Mengembalikan NULL jika tidak
ditemukan
}
```

Telkom University Purwokerto

main.cpp

```cpp
#include <iostream>
#include "singglelist.h"

using namespace std;

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

    createList(L);

    // Alokasi dan memasukkan nilai sesuai urutan
dalam ilustrasi
    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);

    // Mencetak informasi dari List
    printInfo(L);

    // Mencari elemen dengan info 8
    address found = findElm(L, 8);
    if (found != NULL) {
        cout << "Elemen dengan info 8 ditemukan."
<< endl;
    } else {
        cout << "Elemen dengan info 8 tidak
ditemukan." << endl;
    }

    return 0;
}
```

Output:

```
8 ditemukan dalam list.
```

4.3 Program:

singlelist.h

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Prosedur dan fungsi ADT Single Linked List
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertFirst(List &L, address P);
address findElm(const List &L, infotype x);
int sumInfo(const List &L); // Deklarasi fungsi
sumInfo

#endif
```

Telkom University Purwokerto

singlelist.cpp

```cpp
#include <iostream>
#include "singlelist.h"

using namespace std;

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

// Fungsi untuk mengalokasi node baru dengan nilai x
address alokasi(infotype x) {
    address P = new ElmList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

// Prosedur untuk mendealokasi node
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Prosedur untuk mencetak info dari list
void printInfo(const List &L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

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

// Fungsi untuk mencari elemen dengan nilai x dalam list
address findElm(const List &L, infotype x) {
    address P = L.First;
    while (P != NULL) {
        if (P->info == x) {
            return P; // Mengembalikan alamat node jika ditemukan
        }
        P = P->next;
    }
    return NULL; // Mengembalikan NULL jika tidak ditemukan
}

// Fungsi untuk menghitung jumlah total info dari seluruh elemen
int sumInfo(const List &L) {
    int sum = 0;
    address P = L.First;
    while (P != NULL) {
        sum += P->info; // Menambahkan nilai info setiap elemen ke sum
        P = P->next;
    }
    return sum;
}
```

Telkom University Purwokerto

main.cpp

```cpp
#include <iostream>
#include "singlelist.h"

using namespace std;

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

    createList(L);

    // Alokasi dan memasukkan nilai sesuai urutan
dalam ilustrasi
    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 jumlah total info
    int total = sumInfo(L);
    cout << "Total info dari kelima elemen adalah
" << total << endl;

    return 0;
}
```

Output:

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

5. Kesimpulan

Implementasi *single linked list* memungkinkan kita memahami penggunaan *pointer* dan operasi dasar seperti *searching*, *insertion*, *deletion*, dan *update*. Dengan menggunakan *Abstract Data Type* (ADT) yang dipisahkan dalam file *.c* dan *.h*, kode menjadi lebih modular dan mudah dikelola. ADT pada *linked list* menyederhanakan pemrograman dengan menyembunyikan detail implementasi, sehingga mempermudah penggunaan dan meningkatkan efisiensi kode.