I. COVER

LAPORAN PRAKTIKUM

Modul 4

SINGLE LINKED LIST Bagian 2

Disusun Oleh:

Haza Zaidan Zidna Fann

(2311104056)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

II. 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

III. LANDASAN TEORI

Searching adalah operasi dasar dalam pengelolaan list yang melibatkan pencarian terhadap node tertentu. Proses ini dilakukan dengan mengunjungi setiap node dalam list dan berhenti ketika node yang dicari ditemukan. Dengan melakukan operasi searching, berbagai operasi lain seperti insert after, delete after, dan update menjadi lebih mudah dan efisien.

Proses Searching :

Mengunjungi Node: Proses dimulai dengan mengunjungi node pertama dalam list.

Pencocokan: Setiap node diperiksa untuk melihat apakah itu adalah node yang dicari.

Penghentian: Jika node yang dicari ditemukan, proses pencarian dihentikan.

Manfaat Searching :

Melalui operasi searching, kita dapat melakukan beberapa hal berikut:

Insert After: Setelah menemukan node yang sesuai, kita dapat dengan mudah menambahkan node baru setelahnya.

Delete After: Kita juga dapat menghapus node setelah menemukan posisi yang tepat.

Update: Mengupdate informasi pada node yang sudah ada menjadi lebih sederhana karena kita sudah mengetahui lokasi node tersebut.

IV. GUIDED

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

struct Node {
    int data;
    Node* next;
};

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;
    }
}

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;
    }
}

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;
}

void display(Node* node) {
    while (node != nullptr) {
        cout << node->data << " ";
        node = node->next;
    }
    cout << endl;
}

void deleteElement(Node*& head, int x) {
    if (head == nullptr) {
        cout << "Linked list kosong" << endl;
        return;
    }

    // Jika elemen pertama adalah yang ingin dihapus
    if (head->data == x) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    // Cari elemen yang ingin dihapus
    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;
    }

    cout << "Elemen tidak ditemukan dalam linked list" << endl;
}

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 list: ";
    display(head);

    int x;
    cout << "Masukan 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;
}
```

```
Elemen dalam linked list: 18 14 11 7 5 3
Masukan elemen yang ingin dicari: 11
Elemen ditemukan pada indeks 2
Masukan elemen yang ingin dihapus: 14
Elemen dalam linked list setelah penghapusan: 18 11 7 5 3
```

1. **Struktur Node**:

   Didefinisikan struktur Node yang memiliki dua atribut: data (untuk menyimpan nilai) dan next (pointer ke node berikutnya).

2. **Fungsi insertFirst**:

   Menyisipkan node baru di awal linked list.

   Jika tail kosong, maka node baru juga menjadi tail.

3. **Fungsi insertLast**:

   Menyisipkan node baru di akhir linked list (meskipun implementasinya sama dengan insertFirst, seharusnya mengupdate pointer next dari node terakhir).

4. **Fungsi findElement**:

   Mencari elemen dengan nilai tertentu (x) dalam linked list.

   Mengembalikan indeks jika ditemukan, atau -1 jika tidak.

5. **Fungsi display**:

   Menampilkan semua elemen dalam linked list.

6. **Fungsi deleteElement**:

   Menghapus elemen dengan nilai tertentu (x) dari linked list.

   Jika elemen pertama yang dihapus, pointer head diperbarui.

   Jika elemen tidak ditemukan, menampilkan pesan.

7. **Fungsi main**:

   Inisialisasi head dan tail sebagai null.

   Menyisipkan beberapa elemen menggunakan insertFirst.

   Menampilkan elemen dalam linked list.

   Mencari elemen berdasarkan input pengguna dan menampilkan hasilnya.

   Menghapus elemen berdasarkan input pengguna dan menampilkan daftar setelah penghapusan.

Alur Program :

Node: Struktur data dengan data (nilai) dan next (pointer).

insertFirst: Tambah node di awal. Update head dan tail jika list kosong.

insertLast: Tambah node di akhir. Update tail dan pointer next dari node terakhir.

findElement: Cari nilai tertentu. Kembalikan indeks atau -1 jika tidak ditemukan.

display: Cetak semua elemen dalam list.

deleteElement: Hapus node dengan nilai tertentu. Update pointer head, next, atau tail.

Main: Buat list, tambahkan elemen, cari elemen, hapus elemen, dan tampilkan hasilnya.

V. UNGUIDED

```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;
};

// Function prototypes
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertFirst(List &L, address P);
address findElm(List L, infotype x);
int sumInfo(List L);

#endif
```

Tipe Data:

infotype: Alias untuk int.

address: Pointer ke elmlist.

Struktur Data:

elmlist: Menyimpan info dan pointer next.

List: Menunjuk ke elemen pertama (first).

Fungsi:

createList(): Buat daftar kosong.

alokasi(): Alokasikan elemen baru.

dealokasi(): Hapus elemen dan bebaskan memori.

printInfo(): Cetak semua elemen.

insertFirst(): Tambah elemen di awal.

findElm(): Cari elemen dengan nilai tertentu.

sumInfo(): Hitung total nilai elemen.

Inisialisasi Daftar (createList):

Membuat daftar kosong dengan mengatur first pada List ke nullptr.

Alokasi Memori untuk Elemen Baru (alokasi):

Menciptakan elemen baru (elmlist) dengan nilai info tertentu dan mengatur pointer next ke nullptr.

Dealokasi Memori (dealokasi):

Membebaskan memori dari elemen tertentu untuk menghindari kebocoran memori.

Menampilkan Daftar (printInfo):

Melintasi semua elemen dalam daftar dan menampilkan nilai info dari setiap elemen.

Menambah Elemen di Awal Daftar (insertFirst):

Menambahkan elemen baru di awal daftar dengan mengubah pointer next elemen baru ke elemen yang saat ini pertama, lalu memperbarui first untuk menunjuk ke elemen baru.

Mencari Elemen dalam Daftar (findElm):

Melakukan pencarian elemen dengan nilai tertentu (info). Jika ditemukan, mengembalikan alamat elemen tersebut, jika tidak, mengembalikan nullptr.

Menjumlahkan Nilai Elemen (sumInfo):

Menghitung dan mengembalikan jumlah total nilai info dari semua elemen dalam daftar.

```cpp
#include "singlelist.h"
// Fungsi untuk memeriksa apakah list kosong
bool ListEmpty(list L) {
    return (L.first == Nil);
}

// Fungsi untuk membuat list kosong
void CreateList(list &L) {
    L.first = Nil;
}

// Fungsi untuk alokasi elemen baru
address alokasi(infotype x) {
    address P = new elmlist;
    P->info = x;
    P->next = Nil;
    return P;
}

// Fungsi untuk mencari elemen dengan info tertentu
address findElm(list L, infotype X) {
    address P = L.first;
    while (P != Nil) {
        if (P->info == X) {
            return P; // Mengembalikan alamat elemen jika ditemukan
        }
        P = P->next;
    }
    return Nil; // Mengembalikan Nil jika tidak ditemukan
}

// Fungsi untuk menghitung total nilai info dari semua elemen
int sumInfo(list L) {
    int sum = 0;
    address P = L.first;
    while (P != Nil) {
        sum += P->info;
        P = P->next;
    }
    return sum;
}

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

// Fungsi untuk mencetak semua elemen
void printInfo(list L) {
    address P = L.first;
    while (P != Nil) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

```

ListEmpty(): Cek apakah daftar kosong.

CreateList(): Inisialisasi daftar kosong.

alokasi(): Buat elemen baru dan kembalikan alamatnya.

findElm(): Cari elemen dengan nilai tertentu.

sumInfo(): Hitung total nilai elemen.

insertFirst(): Tambah elemen di awal.

printInfo(): Cetak semua elemen.

Inisialisasi Daftar:

Gunakan CreateList() untuk membuat daftar baru.

Memeriksa Apakah Daftar Kosong:

Gunakan ListEmpty() untuk mengecek kondisi daftar.

Menambah Elemen:

Gunakan alokasi() untuk membuat elemen baru.

Gunakan insertFirst() untuk menambah elemen di awal.

Mencari Elemen:

Gunakan findElm() untuk mencari elemen dengan nilai tertentu.

Menjumlahkan Nilai Elemen:

Gunakan sumInfo() untuk menghitung total nilai dari semua elemen.

Mencetak Elemen:

Gunakan printInfo() untuk menampilkan semua elemen dalam daftar.

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

using namespace std;

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;
}
```

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

using namespace std;

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);

    address foundElement = findElm(L, 8);
    if (foundElement != NULL) {
        cout << "8 ditemukan dalam list" << endl;
    } else {
        cout << "8 tidak ditemukan dalam list" << endl;
    }

    return 0;
}
```

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

using namespace std;

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);

    int totalInfo = sumInfo(L);
    cout << "Total info dari kelima elemen adalah " << totalInfo << endl;

    return 0;
}
```

Program Pertama:

1. Inisialisasi daftar kosong (CreateList).
2. Alokasikan lima elemen dengan nilai 2, 0, 8, 12, dan 9.
3. Tambahkan elemen tersebut ke awal daftar (insertFirst).
4. Cetak semua elemen dalam daftar (printInfo).

Program Kedua:

1. Inisialisasi daftar kosong (CreateList).
2. Alokasikan lima elemen dengan nilai 2, 0, 8, 12, dan 9.
3. Tambahkan elemen tersebut ke awal daftar (insertFirst).
4. Cari elemen dengan nilai 8 (findElm).
5. Tampilkan pesan jika elemen ditemukan atau tidak.

Program Ketiga:

1. Inisialisasi daftar kosong (CreateList).
2. Alokasikan lima elemen dengan nilai 2, 0, 8, 12, dan 9.
3. Tambahkan elemen tersebut ke awal daftar (insertFirst).
4. Hitung total nilai semua elemen dalam daftar (sumInfo).
5. Cetak total nilai tersebut.

VI. KESIMPULAN

Searching adalah operasi dasar pada list yang melibatkan pencarian node tertentu dengan mengunjungi setiap node hingga ditemukan. Ini memudahkan operasi lanjutan seperti menambah (insert after), menghapus (delete after), dan memperbarui (update) node karena posisi node yang dicari sudah diketahui**.**