LAPORAN PRAKTIKUM
Modul 5
“SINGLE LINKED LIST (BAGIAN KEDUA)”

Disusun Oleh:
Benedictus Qsota Noventino Baru - 2311104029
S1SE07A

Dosen :
Yudha Islami Sulistya, S.Kom., M.Cs

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM
PURWOKERTO

2024

LATIHAN MODUL (3 soal)

1. Membuat ADT Single Linked list di dalam file “singlelist.h”

a. File 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
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertFirst(List &L, address P);

#endif
```

Penjelasan :

1. **Header Guard**: Mencegah duplikasi saat file di-*include* berulang kali.
2. **Tipe Data**:
   - `infotype`: Alias untuk `int` sebagai data elemen.
   - `address`: Pointer ke elemen list (`Elmlist`).

1. **Struktur Data**:
   - **`Elmlist`**: Node dengan data (`info`) dan pointer ke elemen berikutnya (`next`).
   - **`List`**: Menyimpan pointer ke elemen pertama (`first`).

1. **Fungsi/Prosedur Utama**:
   - **`createList`**: Inisialisasi list kosong.
   - **`alokasi`**: Alokasi memori untuk elemen baru.
   - **`dealokasi`**: Membebaskan memori elemen.

- **`printInfo`**: Menampilkan seluruh elemen list.
- **`insertFirst`**: Menambahkan elemen di awal list.

b. File singlelist.cpp

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

// Membuat list kosong
void createList(List &L) {
    L.first = NULL;
}

// Mengalokasikan memori untuk elemen baru
address alokasi(infotype x) {
    address P = new Elmlist;
    P->info = x;
    P->next = NULL;
    return P;
}

// Mengembalikan memori elemen ke sistem
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Menambahkan elemen baru di awal list
void insertFirst(List &L, address P) {
    P->next = L.first;
    L.first = P;
}

// Mencetak semua elemen di list
void printInfo(List L) {
    address P = L.first;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}
```

Penjelasan :

`createList`

- Menginisialisasi list kosong dengan menyetel `L.first = NULL`.
- Artinya, list belum memiliki elemen apa pun.

`alokasi`

- Membuat node baru dengan **nilai info** dari parameter `x`.
- Node baru tersebut diinisialisasi dengan **pointer `next` bernilai `NULL`**.

`dealokasi`

- **Menghapus node** (elemen) dari memori menggunakan `delete`.
- Setelah dihapus, pointer diatur kembali menjadi `NULL` untuk mencegah dangling pointer.

`insertFirst`

- **Menambahkan elemen baru di awal list**.
- Node baru (`P`) menunjuk ke elemen yang sebelumnya berada di awal, lalu `L.first` diperbarui ke `P`.

`printInfo`

- **Mencetak semua elemen** yang ada di dalam list satu per satu hingga mencapai `NULL` (akhir list).
- Setiap elemen dipisahkan dengan spasi.

c. File main.cpp

```cpp
#include <iostream>
#include "singlelist.h"
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);  // Output harus sesuai dengan contoh: 9 12 8 0 2

    return 0;
}
```

Struktur Node:

- Setiap elemen dalam linked list diwakili oleh struktur yang disebut `Node`, yang memiliki dua bagian:
    - `data`: Menyimpan nilai (misalnya, integer).
    - `next`: Pointer yang menunjuk ke node berikutnya dalam list.

Struktur List:

- Struktur ini memiliki satu anggota, yaitu pointer `first`, yang menunjuk ke node pertama dalam linked list. Jika list kosong, `first` akan bernilai `nullptr`.

Fungsi `createList`:

- Menginisialisasi list dengan mengatur pointer `first` menjadi `nullptr`, menandakan bahwa list kosong.

Fungsi `alokasi`:

- Menciptakan dan mengalokasikan memori untuk node baru. Fungsi ini mengembalikan pointer ke node yang baru dibuat.

Fungsi `insertFirst`:

- Menyisipkan node baru di awal list. Node baru menjadi node pertama dan pointer `next` dari node baru diatur untuk menunjuk ke node sebelumnya yang menjadi kedua.

Fungsi `printInfo`:

- Mencetak semua nilai dalam list, dimulai dari node pertama hingga node terakhir, memisahkan nilai dengan spasi.

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

```c
#include <stdio.h>
#include <stdlib.h>

// Definisi struktur Node untuk linked list
typedef struct Node {
    int info;
    struct Node* next;
} Node;

typedef Node* List;      // Alias untuk pointer ke Node
typedef Node* address;   // Alias untuk alamat Node

// Fungsi untuk mencari elemen dengan info tertentu
address findElm(List L, int x) {
    address current = L;
    while (current != NULL) {
        if (current->info == x) {
            return current;  // Elemen ditemukan
        }
        current = current->next;
    }
    return NULL;  // Jika tidak ditemukan
}

// Fungsi untuk menambahkan elemen baru di awal list
void insertFirst(List* L, int info) {
    address newNode = (address)malloc(sizeof(Node));
    newNode->info = info;
    newNode->next = *L;
    *L = newNode;
}

int main() {
    List L = NULL;  // Inisialisasi list kosong

    // Tambahkan elemen ke dalam list
    insertFirst(&L, 2);
    insertFirst(&L, 0);
    insertFirst(&L, 8);
    insertFirst(&L, 12);
    insertFirst(&L, 9);

    // Mencari elemen dengan info 8
    int target = 8;
    address result = findElm(L, target);

    if (result != NULL) {
        printf("%d ditemukan dalam list\n", target);
    } else {
        printf("%d tidak ditemukan dalam list\n", target);
    }

    return 0;
}
```

Struktur Node:

- Struktur `Node` mendefinisikan elemen dari linked list. Setiap node memiliki:
  - `info`: Menyimpan data integer.
  - `next`: Pointer yang menunjuk ke node berikutnya.

Tipe Data:

- `List` dan `address` adalah alias untuk pointer ke `Node`, membuatnya lebih mudah untuk merujuk ke jenis data ini di seluruh program.

Fungsi `findElm`:

- Mencari elemen dalam linked list dengan informasi yang sesuai dengan nilai yang diberikan (`x`).
- Melalui loop, fungsi ini memeriksa setiap node untuk menemukan node dengan `info` yang sesuai.
- Mengembalikan pointer ke node jika ditemukan, atau `NULL` jika tidak ditemukan.

Fungsi `insertFirst`:

- Menambahkan elemen baru ke awal linked list.
- Mengalokasikan memori untuk node baru, menginisialisasi `info`, dan mengatur `next` dari node baru untuk menunjuk ke node yang ada di list.
- Mengupdate pointer list (`L`) untuk menunjuk ke node baru.

Fungsi `main`:

- Menginisialisasi linked list sebagai kosong (`L = NULL`).
- Menambahkan beberapa elemen ke list dengan memanggil `insertFirst`.
- Mencari elemen dengan nilai `8` menggunakan `findElm`.
- Mencetak hasil pencarian, menunjukkan apakah elemen tersebut ditemukan dalam list atau tidak.

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

```c
#include <stdio.h>
#include <stdlib.h>

// Definisi struktur Node untuk linked list
typedef struct Node {
    int info;
    struct Node* next;
} Node;

typedef Node* List;
typedef Node* address;

// Fungsi untuk mencari elemen dengan nilai tertentu
address findElm(List L, int x) {
    address current = L;
    while (current != NULL) {
        if (current->info == x) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

// Fungsi untuk menambahkan elemen baru di awal list
void insertFirst(List* L, int info) {
    address newNode = (address)malloc(sizeof(Node));
    newNode->info = info;
    newNode->next = *L;
    *L = newNode;
}

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

int main() {
    List L = NULL;  // Inisialisasi list kosong

    // Menambahkan elemen ke dalam list
    insertFirst(&L, 2);
    insertFirst(&L, 0);
    insertFirst(&L, 8);
    insertFirst(&L, 12);
    insertFirst(&L, 9);

    // Menghitung total info seluruh elemen
    int total = sumInfo(L);
    printf("Total info dari kelima elemen adalah %d\n", total);

    return 0;
}
```

Struktur Node:

- Struktur `Node` mendefinisikan elemen dari linked list. Setiap node memiliki:
  - `info`: Menyimpan data bertipe integer.
  - `next`: Pointer yang menunjuk ke node berikutnya dalam list.

Tipe Data:

- `List` dan `address` adalah alias untuk pointer ke `Node`, membuat kode lebih mudah dibaca dan dikelola.

Fungsi `findElm`:

- Mencari elemen dalam linked list yang memiliki nilai tertentu (`x`).
- Menggunakan loop untuk memeriksa setiap node hingga menemukan yang sesuai atau mencapai akhir list.
- Mengembalikan pointer ke node yang ditemukan, atau `NULL` jika tidak ditemukan.

Fungsi `insertFirst`:

- Menambahkan elemen baru di awal linked list.
- Mengalokasikan memori untuk node baru, menginisialisasi `info`, dan mengatur `next` dari node baru untuk menunjuk ke node yang ada di list.
- Mengupdate pointer list (`L`) untuk menunjuk ke node baru.

Fungsi `sumInfo`:

- Menghitung total dari semua nilai `info` dalam linked list.
- Menggunakan loop untuk menjelajahi setiap node, menambahkan nilai `info` dari setiap node ke variabel `sum`.
- Mengembalikan total sum setelah semua elemen telah diperiksa.

Fungsi `main`:

- Menginisialisasi linked list sebagai kosong (`L = NULL`).
- Menambahkan beberapa elemen ke list dengan memanggil `insertFirst`.
- Menghitung total nilai dari semua elemen dalam list menggunakan `sumInfo`.
- Mencetak hasil total ke konsol.

UNGUIDED

Soal 1

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

// Definisi struktur Node untuk linked list
struct Node {
    int info;
    Node* next;
};

// Tipe List sebagai pointer ke Node
typedef Node* List;

// Fungsi untuk menambahkan elemen baru di awal list
void insertFirst(List& L, int info) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->info = info;
    newNode->next = L;
    L = newNode;
}

// Fungsi untuk mencari elemen dengan nilai tertentu
void searchElement(List L, int i) {
    Node* current = L;
    int position = 1;  // Inisialisasi posisi

    // Melakukan perulangan untuk mencari elemen
    while (current != nullptr) {
        if (current->info == i) {
            cout << "Elemen " << i << " ditemukan pada posisi ke " << position
                 << ", alamat: " << current << endl;
            return;  // Keluar dari fungsi setelah menemukan elemen
        }
        current = current->next;  // Pindah ke node berikutnya
        position++;  // Increment posisi
    }

    // Jika elemen tidak ditemukan
    cout << "Elemen " << i << " tidak ditemukan dalam list." << endl;
}

int main() {
    List L = nullptr;  // Inisialisasi list kosong
    int input;

    // Minta pengguna memasukkan 6 elemen
    cout << "Masukkan 6 elemen integer ke dalam list:" << endl;
    for (int j = 0; j < 6; j++) {
        cout << "Elemen " << (j + 1) << ": ";
        cin >> input;
        insertFirst(L, input);  // Tambahkan elemen ke list
    }

    // Minta pengguna memasukkan nilai yang ingin dicari
    cout << "Masukkan nilai yang ingin dicari: ";
    cin >> input;

    // Panggil fungsi untuk mencari elemen
    searchElement(L, input);

    // Pembebasan memori (opsional)
    // Anda dapat menambahkan kode untuk membebaskan memori jika diperlukan

    return 0;
}
```

Soal 2

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

// Definisi struktur Node untuk linked list
struct Node {
    int info;
    Node* next;
};

// Tipe List sebagai pointer ke Node
typedef Node* List;

// Fungsi untuk menambahkan elemen baru di awal list
void insertFirst(List& L, int info) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->info = info;
    newNode->next = L;
    L = newNode;
}

// Prosedur untuk menampilkan elemen-elemen dalam list
void printList(List L) {
    Node* current = L;
    while (current != nullptr) {
        cout << current->info << " ";
        current = current->next;
    }
    cout << endl;
}

// Prosedur untuk mengurutkan list menggunakan Bubble Sort
void bubbleSortList(List& L) {
    if (L == nullptr || L->next == nullptr) return;  // Jika list kosong atau hanya ada satu elemen

    bool swapped;
    do {
        swapped = false;  // Set swapped ke false di awal iterasi
        Node* current = L;

        while (current != nullptr && current->next != nullptr) {
            if (current->info > current->next->info) {
                // Tukar data antara current dan current->next
                swap(current->info, current->next->info);
                swapped = true;  // Set swapped menjadi true karena terjadi pertukaran
            }
            current = current->next;  // Pindah ke node berikutnya
        }
    } while (swapped);  // Ulangi sampai tidak ada lagi pertukaran yang dilakukan
}

int main() {
    List L = nullptr;  // Inisialisasi list kosong
    int input;

    // Minta pengguna memasukkan 5 elemen
    cout << "Masukkan 5 elemen integer ke dalam list:" << endl;
    for (int j = 0; j < 5; j++) {
        cout << "Elemen " << (j + 1) << ": ";
        cin >> input;
        insertFirst(L, input);  // Tambahkan elemen ke list
    }

    cout << "List sebelum diurutkan: ";
    printList(L);  // Tampilkan list sebelum diurutkan

    // Mengurutkan list menggunakan Bubble Sort
    bubbleSortList(L);

    cout << "List setelah diurutkan: ";
    printList(L);  // Tampilkan list setelah diurutkan

    return 0;
}
```

Soal 3

```cpp
int main() {
    List L = nullptr; // Inisialisasi list kosong
    int input;

    // Minta pengguna memasukkan 4 elemen
    cout << "Masukkan 4 elemen integer ke dalam list (harus terurut):" << endl;
    for (int j = 0; j < 4; j++) {
        cout << "Elemen " << (j + 1) << ": ";
        cin >> input;
        // Hanya mengizinkan input terurut
        if (L == nullptr || (L != nullptr && input >= L->info)) {
            Node* newNode = createNode(input);
            insertSorted(L, newNode);
        } else {
            cout << "Input harus terurut. Silakan coba lagi." << endl;
            j--; // Mengurangi j agar pengguna bisa menginput ulang
        }
    }

    // Tampilkan list setelah elemen dimasukkan
    cout << "List sebelum menambahkan elemen baru: ";
    printList(L);

    // Minta pengguna memasukkan elemen tambahan
    cout << "Masukkan elemen tambahan untuk dimasukkan ke list: ";
    cin >> input;
    Node* newNode = createNode(input);
    insertSorted(L, newNode); // Menambahkan elemen baru ke list

    // Tampilkan list setelah elemen baru dimasukkan
    cout << "List setelah menambahkan elemen baru: ";
    printList(L);

    return 0;
}
```