LAPORAN PRAKTIKUM

Modul 5

DAFTAR BERANTAI TUNGGAL (*SINGLE LINKED LIST*) BAGIAN 2

Disusun Oleh:

Adhiansyah Muhammad Pradana Farawowan - 2211104038

S1SE-07-02

Asisten Praktikum:

Aldi Putra

Andini Nur Hidayah

Dosen:

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

PROGRAM STUDI S1 REKAYASAN PERANGKAT LUNAK

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM PURWOKERTO

2024

A. Tujuan

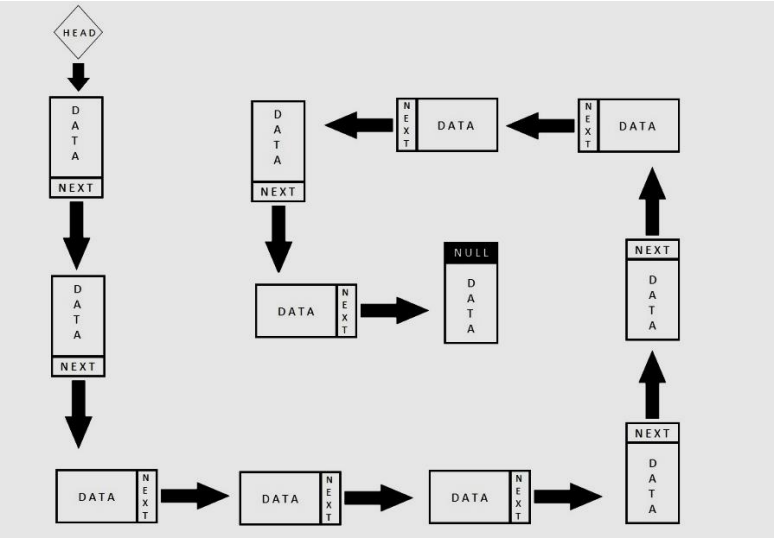
Laporan praktikum ini bertujuan untuk memperkenalkan, memahami, mengelola, dan menuntun cara mengimplementasikan daftar berantai tunggal.

B. Landasan Teori

a. Daftar berantai

Daftar berantai (*linked list*) adalah struktur data berupa koleksi yang elemennya menunjuk pada elemen lain yang sedemikian sehingga membentuk sebuah urutan yang berantai.

Sebuah elemen daftar terdiri dari isi data/informasi itu sendiri dan sebuah tipe yang menunjuk ke elemen selanjutnya (*next*). Dalam C++, *next* ini bisa diimplementasikan dengan penunjuk (*pointer*).



Ilustrasi daftar berantai. Sumber: Vhcomptech di commonswiki.

Daftar berantai memiliki dua jenis, yaitu tunggal dan ganda. Daftar berantai ganda memiliki penunjuk ke elemen sebelumnya dan setelahnya. Laporan ini akan membahas daftar berantai tunggal.

C. Bimbingan (*guided*)

Bimbingan hari ini adalah mengimplementasikan daftar berantai beserta mekanisme pencarian elemen, dengan modifikasi sendiri dari penulis.

```
#include <iostream>

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


// Buat daftar kosong
```

```
std::nullptr_t empty_element()
{
    return nullptr;
}

// Sisip elemen ke awal daftar
void insert_as_first(NODE *&head, NODE *&tail, int new_data)
{
    NODE *new_element = new NODE;
    new_element->data = new_data;

    // Buat lanjutan dari elemen new_element ke elemen awal sebelumnya
    new_element->next = head;

    // Jadikan elemen new_element sebagai head
    head = new_element;

    // Sekalian kalau tail ternyata kosong
    if (tail == empty_element())
    {
        tail = new_element;
    }
}

// Sisip elemen ke akhir dafar
void insert_as_last(NODE *&head, NODE *&tail, int new_data)
{
    NODE *new_element = new NODE;
    new_element->data = new_data;
    new_element->next = empty_element();

    if (head == empty_element())
    {
        head = new_element;
        tail = new_element;
    }
    else
    {
        // Buat lanjutan dari tail ke new_element
        tail->next = new_element;

        // Jadikan new_element sebagai tail
        tail = new_element;
    }
}

// Cari elemen
int find_element(NODE *head, int searched_data)
{
    NODE *current = head;
    int position = 0;

    while (current != empty_element())
    {
        if (current->data == searched_data)
        {
            return position;
        }

        current = current->next;
        position = position + 1;
    }
```

```
    return -1;
}

void delete_element(NODE *&head, int searched_data)
{
    if (head == empty_element())
    {
        std::cout << "The list is empty." << '\n';
        return;
    }

    // Jika head adalah yang dicari
    if (head->data == searched_data)
    {
        NODE *temp = head;
        head = head->next;
        delete temp;
        return;
    }

    NODE *current = head;
    while (current->next != empty_element())
    {
        if (current->next->data == searched_data)
        {
            NODE *temp = current->next;
            current->next = temp->next;
            delete temp;
            return;
        }

        current = current->next;
    }
}

// Cetak daftar
void print_list(NODE *element)
{
    while (element != empty_element())
    {
        std::cout << element->data << " ";
        element = element->next;
    }

    std::cout << '\n';
}

int main() {
    NODE* head = empty_element();
    NODE* tail = empty_element();

    insert_as_first(head, tail, 53);
    insert_as_first(head, tail, 97);
    insert_as_first(head, tail, 41);

    insert_as_last(head, tail, 313);
    insert_as_last(head, tail, 283);
    insert_as_last(head, tail, 9109);

    std::cout << "Elements in list: ";
    print_list(head);

    int x;
```

```
    std::cout << "Search element: ";
    std::cin >> x;

    int result = find_element(head, x);

    if (result >= 0) {
        std::cout << "The data is located at position " << result << '\n';
    } else {
        std::cout << "The data is not found" << '\n';
    }

    std::cout << "Input element you wanted to delete: ";
    std::cin >> x;
    delete_element(head, x);

    std::cout << "Current list: ";
    print_list(head);

    return 0;
}
```

Output

```
>a.exe
Elements in list: 41 97 53 313 283 9109
Search element: 9109
The data is located at position 5
Input element you wanted to delete: 41
Current list: 97 53 313 283 9109
```

D. Tugas mandiri (*unguided)*

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

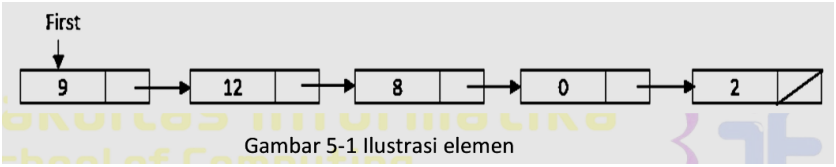
```
Type infotype : int
Type address  : pointer to ElmList

Type ElmList <
    info : infotype
    next : address
>

Type List : < First : address >

prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertFirst( in/out L : List, in P : address )
```

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



Gambar 5-1 Ilustrasi elemen

Cobalah hasil implementasi ADT pada file “main.cpp”

```
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;
}
```

```
9 12 8 0 2

Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Gambar 5-2 *Output* singlelist

singlelist.h

```
#ifndef _SINGLELIST_H
#define _SINGLELIST_H

struct ElmList
{
    int info;
    struct ElmList *next;
};

struct List
{
    struct ElmList *First;
};

void create_list(List &L);

ElmList *alokasi(int x);

void dealokasi(ElmList *P);

void print_info(List &L);

ElmList *find_elm(List &L, int x);

#endif
```

singlelist.cpp

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

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

```
ElmList *alokasi(int x)
{
    ElmList *new_elm = new ElmList;
    new_elm->info = x;
    new_elm->next = nullptr;

    return new_elm;
}

void dealokasi(ElmList *P)
{
    delete P;
}

void print_info(List &L)
{
    if (L.First == nullptr)
    {
        std::cout << "List is empty." << '\n';
    }

    ElmList *current = L.First;
    while (current != nullptr)
    {
        std::cout << current->info << " ";
        current = current->next;
    }
}

void insert_first(List &L, ElmList *P)
{
    P->next = L.First;
    L.First = P;
}

ElmList *find_elm(List &L, int x)
{
    if (L.First == nullptr)
    {
        std::cout << "List is empty." << '\n';
    }

    ElmList *current = L.First;
    while (current != nullptr)
    {
        if (current->info == x)
        {
            return current;
        }
        current = current->next;
    }

    return nullptr;
}
```

main.cpp

```
#include "singlelist.cpp"

int main()
{
    // UNGUIDED 1
    List L;
    ElmList *P1, *P2, *P3, *P4, *P5;
    create_list(L);
    P1 = alokasi(2);
    insert_first(L, P1);
    P2 = alokasi(0);
    insert_first(L, P2);
    P3 = alokasi(8);
    insert_first(L, P3);
    P4 = alokasi(12);
    insert_first(L, P4);
    P5 = alokasi(9);
    insert_first(L, P5);

    print_info(L);
    // UNGUIDED 1

    ...
    return 0;
}
```

Output

```
                                                                >a.exe
9 12 8 0 2 OUTPUT PROGRAM SOAL 1
8 is found in list OUTPUT PROGRAM SOAL 2
Total info from the list is 31 OUTPUT PROGRAM SOAL 3
```

Kode sumber lengkap tersedia di direktori UNGUIDED

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

```
8 ditemukan dalam list
Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

Gambar 5-3 *Output* pencarian 8

main.cpp

```
#include "singlelist.cpp"

int main()
{
    ...
    // UNGUIDED 2
    ElmList* searched_element = find_elm(L, 8);
    if (searched_element != nullptr) {
        std::cout << searched_element->info << " is found in list" << '\n';
    }
    // UNGUIDED 2
    ...

    return 0;
}
```

Output

```
                                                                >a.exe
9 12 8 0 2 OUTPUT PROGRAM SOAL 1
8 is found in list OUTPUT PROGRAM SOAL 2
Total info from the list is 31 OUTPUT PROGRAM SOAL 3
```

Kode sumber lengkap tersedia di direktori UNGUIDED

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

```
Total info dari kelima elemen adalah 31
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Gambar 5-4 *Output* total info elemen

main.cpp

```
#include "singlelist.cpp"

// Baiknya disini aja
int sum_info(List &L) {
    ElmList *current = L.First;
    int count = 0;
    while (current != nullptr)
    {
        count = count + current->info;
        current = current->next;
    }

    return count;
}


int main()
{
    ...
    // UNGUIDED 3
    std::cout << "Total info from the list is " << sum_info(L) << '\n';
    // UNGUIDED 3

    return 0;
}
```

Output

```
                                                                >a.exe
9 12 8 0 2 OUTPUT PROGRAM SOAL 1
8 is found in list OUTPUT PROGRAM SOAL 2
Total info from the list is 31 OUTPUT PROGRAM SOAL 3
```

Kode sumber lengkap tersedia di direktori UNGUIDED