

**LAPORAN PRAKTIKUM**  
**Modul 4**  
**“SINGLE LINKED LIST (BAGIAN PERTAMA)”**



**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 ( 1 soal )

### 1. Membuat ADT Single Linked List

```
#ifndef SINGLELIST_H
#define SINGLELIST_H

#include <iostream>
using namespace std;

typedef int infotype;

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

typedef ElmList* address;

struct List {
    address First;
};

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

#endif
```

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

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

// Alokasi memori untuk elemen baru
address alokasi(infotype x) {
    address P = new ElmList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

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

// Menyisipkan elemen di awal list
void insertFirst(List &L, address P) {
    if (L.First == NULL) {
        L.First = P;
    } else {
        P->next = L.First;
        L.First = P;
    }
}

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

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

    // Membuat list kosong
    createList(L);

    // Alokasi dan memasukkan 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);

    // Mencetak isi list
    printInfo(L);

    return 0;
}
```

### Penjelasan Kode:

- **Header File (`singlelist.h`):**
  - Digunakan untuk mendeklarasikan tipe data, prosedur, dan fungsi yang digunakan dalam implementasi linked list.
- **Implementasi File (`singlelist.cpp`):**
  - `createList`: Menginisialisasi linked list kosong.
  - `alokasi`: Membuat elemen baru dengan mengalokasikan memori untuk elemen tersebut.
  - `dealokasi`: Menghapus elemen dari memori.
  - `insertFirst`: Menyisipkan elemen baru di awal linked list.
  - `printInfo`: Mencetak isi linked list.
- **File Utama (`main.cpp`):**
  - Membuat linked list dan memasukkan lima elemen ke dalamnya (9, 12, 8,

0, 2), lalu mencetak isi linked list.

## Hasil Output:

```
noven@NOVEN MINGW64
$ ./program
9 12 8 0 2
```

## SOAL UNGUIDED ( 3 soal )

### 2. Membuat Single Linked List

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

// Definisi struktur node
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk membuat node baru
Node* createNode(int value) {
    Node* newNode = new Node(); // Alokasi memori untuk node baru
    newNode->data = value;
    newNode->next = nullptr; // Inisialisasi next sebagai null
    return newNode;
}

// Fungsi untuk menambah node di depan
void insertAtFront(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head; // Set next dari node baru ke head
    head = newNode; // Update head ke node baru
}

// Fungsi untuk menambah node di belakang
void insertAtEnd(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) { // Jika linked list kosong
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Menyusuri sampai node terakhir
            temp = temp->next;
        }
        temp->next = newNode; // Hubungkan node terakhir dengan node baru
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " → ";
        }
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr; // Inisialisasi linked list sebagai kosong

    // Tambah node di depan (nilai: 10)
    insertAtFront(head, 10);
    // Tambah node di belakang (nilai: 20)
    insertAtEnd(head, 20);
    // Tambah node di depan (nilai: 5)
    insertAtFront(head, 5);

    // Cetak linked list
    cout << "Isi linked list: ";
    printList(head);

    return 0;
}
```

### Penjelasan:

1. **Struct Node**: Struktur untuk node dari linked list. Setiap node memiliki dua bagian: **data** untuk menyimpan nilai dan **next** sebagai pointer ke node berikutnya.
2. **createNode()**: Fungsi untuk membuat node baru dengan nilai tertentu.
3. **insertAtFront()**: Fungsi untuk menambah node di depan linked list. Node baru akan menjadi head.
4. **insertAtEnd()**: Fungsi untuk menambah node di akhir linked list. Fungsi ini menyusuri linked list hingga node terakhir, kemudian menambah node baru di ujungnya.
5. **printList()**: Fungsi untuk mencetak isi linked list dari node pertama hingga node terakhir.
6. **main()**: Fungsi utama yang menginisialisasi linked list dan melakukan operasi penambahan node serta mencetak linked list.

Output:

```
noven@NOVEN MINGW64 ~/Desktop/Pra
$ ./program_2
Isi linked list: 5 -> 10 -> 20
```

### 3. Menghapus Node pada Linked List

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

// Definisi struktur node
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk membuat node baru
Node* createNode(int value) {
    Node* newNode = new Node(); // Alokasi memori untuk node baru
    newNode->data = value;
    newNode->next = nullptr; // Inisialisasi next sebagai null
    return newNode;
}

// Fungsi untuk menambah node di depan
void insertAtFront(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head; // Set next dari node baru ke head
    head = newNode; // Update head ke node baru
}

// Fungsi untuk menambah node di belakang
void insertAtEnd(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) { // Jika linked list kosong
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Menyuri sampai node terakhir
            temp = temp->next;
        }
        temp->next = newNode; // Hubungkan node terakhir dengan node baru
    }
}

// Fungsi untuk menghapus node dengan nilai tertentu
void deleteNode(Node*& head, int value) {
    // Jika linked list kosong
    if (head == nullptr) {
        cout << "Linked list kosong." << endl;
        return;
    }

    // Jika node pertama yang harus dihapus
    if (head->data == value) {
        Node* temp = head;
        head = head->next; // Update head ke node berikutnya
        delete temp; // Hapus node lama
        cout << "Node dengan nilai " << value << " telah dihapus." << endl;
        return;
    }

    // Menyuri linked list untuk mencari node yang akan dihapus
    Node* temp = head;
    while (temp->next != nullptr && temp->next->data != value) {
        temp = temp->next;
    }

    // Jika node ditemukan
    if (temp->next != nullptr) {
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next; // Menghubungkan node sebelumnya dengan node setelahnya
        delete nodeToDelete; // Hapus node
        cout << "Node dengan nilai " << value << " telah dihapus." << endl;
    } else {
        cout << "Node dengan nilai " << value << " tidak ditemukan." << endl;
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr; // Inisialisasi linked list sebagai kosong

    // Tambah node di depan (nilai: 10)
    insertAtFront(head, 10);
    // Tambah node di belakang (nilai: 20)
    insertAtEnd(head, 20);
    // Tambah node di depan (nilai: 5)
    insertAtFront(head, 5);

    // Cetak linked list sebelum penghapusan
    cout << "Linked list sebelum penghapusan: ";
    printList(head);

    // Hapus node dengan nilai tertentu (nilai: 10)
    deleteNode(head, 10);

    // Cetak linked list setelah penghapusan
    cout << "Linked list setelah penghapusan: ";
    printList(head);

    return 0;
}
```

## Penjelasan:

1. **Struct Node**: Struktur untuk node dari linked list. Setiap node memiliki dua bagian: **data** untuk menyimpan nilai dan **next** sebagai pointer ke node berikutnya.
2. **createNode()**: Fungsi untuk membuat node baru dengan nilai tertentu.
3. **insertAtFront()**: Fungsi untuk menambah node di depan linked list.
4. **insertAtEnd()**: Fungsi untuk menambah node di akhir linked list.
5. **deleteNode()**: Fungsi untuk menghapus node dengan nilai tertentu. Fungsi ini mencari node dengan nilai yang diberikan, lalu menghapusnya dengan memperbarui pointer dari node sebelumnya.
6. **printList()**: Fungsi untuk mencetak isi linked list.
7. **main()**: Fungsi utama yang menginisialisasi linked list, melakukan operasi penambahan node, penghapusan node, dan mencetak linked list.

## Output:

```
noven@NOVEN MINGW64 ~/Desktop/Prak SD/04_Single_L
$ ./program_3
Linked list sebelum penghapusan: 5 -> 10 -> 20
Node dengan nilai 10 telah dihapus.
Linked list setelah penghapusan: 5 -> 20
```

#### 4. adfas

```
#include <iostream>

using namespace std;

// Struktur untuk Node linked list
struct Node {
    int data;
    Node* next;
};

// Kelas LinkedList
class LinkedList {
private:
    Node* head;

public:
    // Konstruktor
    LinkedList() : head(nullptr) {}

    // Menambah node di depan
    void addFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    // Menambah node di belakang
    void addBack(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
            return;
        }

        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    // Mencari node dengan nilai tertentu
    bool search(int value) {
        Node* temp = head;
        while (temp != nullptr) {
            if (temp->data == value) {
                return true; // Nilai ditemukan
            }
            temp = temp->next;
        }
        return false; // Nilai tidak ditemukan
    }

    // Menghitung panjang linked list
    int length() {
        int count = 0;
        Node* temp = head;
        while (temp != nullptr) {
            count++;
            temp = temp->next;
        }
        return count;
    }

    // Destructor untuk membersihkan memori
    ~LinkedList() {
        Node* current = head;
        Node* next;

        while (current != nullptr) {
            next = current->next;
            delete current;
            current = next;
        }
    }
};

int main() {
    LinkedList list;

    // Menambah node
    list.addFront(10);
    list.addBack(20);
    list.addFront(5);

    // Mencari node
    int searchValue = 20;
    if (list.search(searchValue)) {
        cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
    }

    // Menghitung panjang linked list
    cout << "Panjang linked list: " << list.length() << endl;

    return 0;
}
```

## Penjelasan Program:

1. **Struktur Node**: Mewakili elemen di dalam linked list dengan data dan pointer ke node berikutnya.
2. **Kelas LinkedList**: Mengelola operasional linked list dengan:
  - `addFront(int value)`: Menambahkan node baru di depan linked list.
  - `addBack(int value)`: Menambahkan node baru di belakang linked list.
  - `search(int value)`: Mencari apakah nilai tertentu ada dalam linked list.
  - `length()`: Menghitung jumlah node dalam linked list.
  - Destructor untuk membersihkan memori saat objek `LinkedList` dihancurkan.
3. **Fungsi main**: Membuat instance dari `LinkedList`, menambah node, mencari nilai tertentu, dan mencetak panjang linked list.

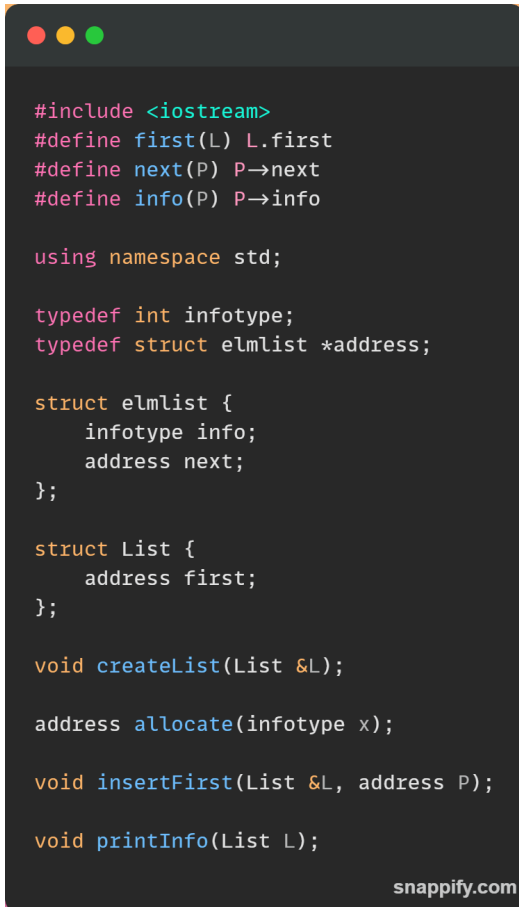
Output :

```
$ ./program_4
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```



## Soal TP

### File list.h

A screenshot of a code editor window with a dark background. The code is written in C++ and defines a linked list structure. It includes a header file, defines macros for accessing list elements, uses the std namespace, defines typedefs for node information and pointers, and declares several functions for list management. The code is color-coded: keywords in blue, identifiers in green, and literals in orange. A watermark 'snappify.com' is visible in the bottom right corner.

```
#include <iostream>
#define first(L) L.first
#define next(P) P->next
#define info(P) P->info

using namespace std;

typedef int infotype;
typedef struct elmList *address;

struct elmList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);

address allocate(infotype x);

void insertFirst(List &L, address P);

void printInfo(List L);
```

### Penjelasan list.h:

- **Makro:** `first(L)`, `next(P)`, dan `info(P)` untuk mempermudah akses ke elemen list dan node.
- **typedef:** Digunakan untuk menyederhanakan tipe data.
- **Struct elmList:** Definisi node dengan `info` dan `next`.
- **Struct List:** Menyimpan alamat node pertama.
- **Fungsi-fungsi:** Deklarasi fungsi dasar seperti `createList`, `allocate`, `insertFirst`, dan `printInfo`.

## File list.cpp

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

// Implementasi fungsi-fungsi List akan ditambahkan di sini
void createList(List &L) {
    /** this procedure will initialize the list L */
    first(L) = NULL;
}

address allocate(infotype x) {
    address P = new elmList;
    info(P) = x;
    next(P) = NULL;
    return P;
}

void insertFirst(List &L, address P) {
    /** TODO: Insert the new element pointed by P to the first of list L */
    next(P) = first(L);
    first(L) = P;
}

void printInfo(List L) {
    /** this procedure will output the info of each element in list L */
    address P = first(L);
    while (P != NULL) {
        cout << info(P) << " ";
        P = next(P);
    }
    cout << endl;
}
```

snappify.com

## Penjelasan list.cpp:

- **createList**: Menginisialisasi list agar kosong dengan `first(L) = NULL`.
- **allocate**: Membuat node baru dengan nilai tertentu.
- **insertFirst**: Menyisipkan node baru di awal list.
- **printInfo**: Mencetak elemen-elemen dalam list secara berurutan.

File main.cpp

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

using namespace std;

int main() {
    List L;
    createList(L);

    address P;

    // Insert 3 elements with 3-digit NIM
    P = allocate(111);
    insertFirst(L, P);

    P = allocate(123);
    insertFirst(L, P);

    P = allocate(147);
    insertFirst(L, P);

    // Print the list
    printInfo(L);

    return 0;
}
```

snappify.com

### Penjelasan main.cpp:

1. **Deklarasi list:** Membuat objek `L` bertipe `List`.
2. **`createList(L)`:** Menginisialisasi list agar kosong.
3. **`allocate`:** Membuat 3 node baru dengan nilai **111, 123, dan 147**.
4. **`insertFirst`:** Setiap node disisipkan di awal list.
5. **`printInfo`:** Mencetak semua elemen dalam list (output: **147 123 111**).

Output :

```
noven@NOVEN MINGW64 ~
$ ./program
147 123 111
```