LAPORAN PRAKTIKUM

PERTEMUAN 4

SINGLE LINKED LIST (BAGIAN PERTAMA)

Nama :

Resita Istania Purwanto (2311104037)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

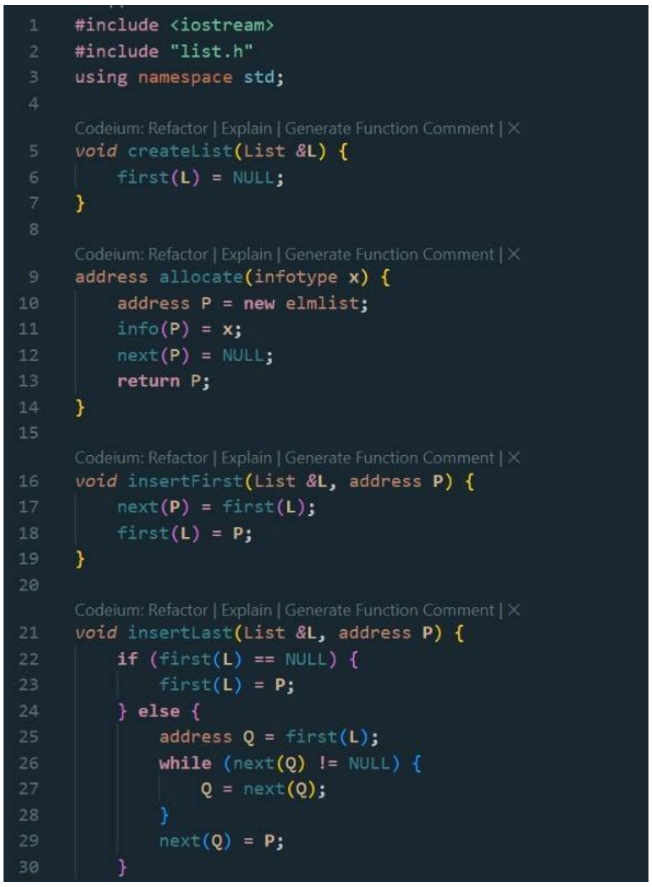
TELKOM UNIVERSITY PURWOKERTO

2024

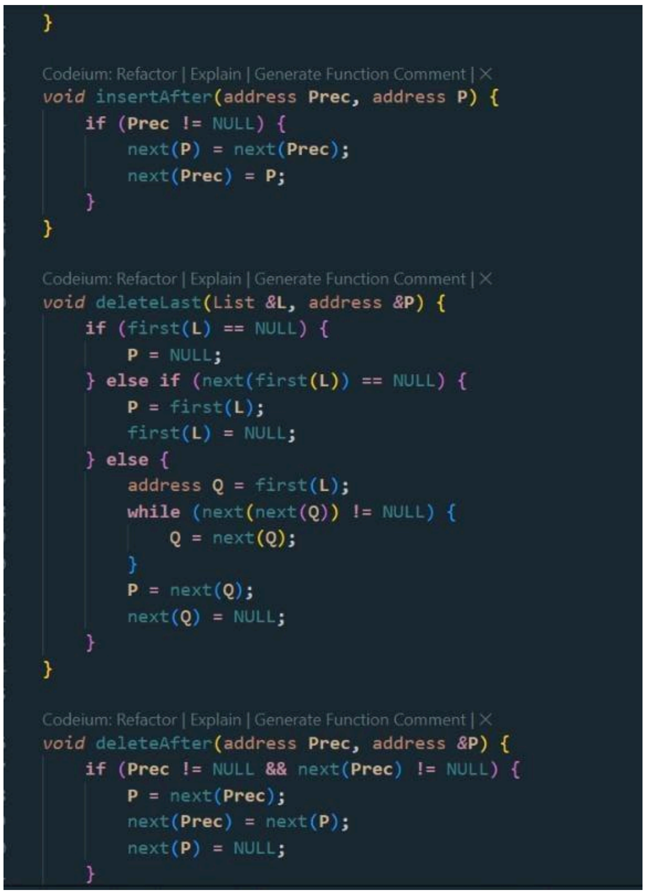
I. TP

Kode:

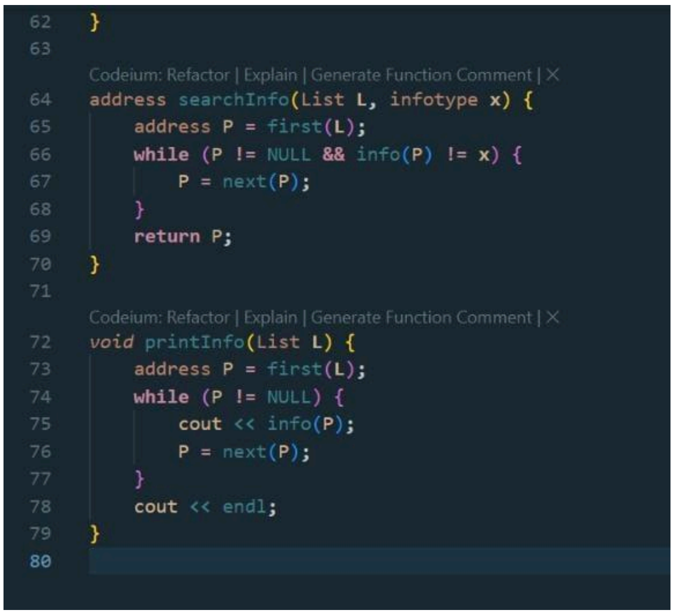
list.h



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

Codeium: Refactor | Explain | Generate Function Comment | ×
void createList(List &L) {
    first(L) = NULL;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
address allocate(infotype x) {
    address P = new elmlist;
    info(P) = x;
    next(P) = NULL;
    return P;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertFirst(List &L, address P) {
    next(P) = first(L);
    first(L) = P;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertLast(List &L, address P) {
    if (first(L) == NULL) {
        first(L) = P;
    } else {
        address Q = first(L);
        while (next(Q) != NULL) {
            Q = next(Q);
        }
        next(Q) = P;
    }
```



```
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertAfter(address Prec, address P) {
    if (Prec != NULL) {
        next(P) = next(Prec);
        next(Prec) = P;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void deleteLast(List &L, address &P) {
    if (first(L) == NULL) {
        P = NULL;
    } else if (next(first(L)) == NULL) {
        P = first(L);
        first(L) = NULL;
    } else {
        address Q = first(L);
        while (next(next(Q)) != NULL) {
            Q = next(Q);
        }
        P = next(Q);
        next(Q) = NULL;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void deleteAfter(address Prec, address &P) {
    if (Prec != NULL && next(Prec) != NULL) {
        P = next(Prec);
        next(Prec) = next(P);
        next(P) = NULL;
    }
```

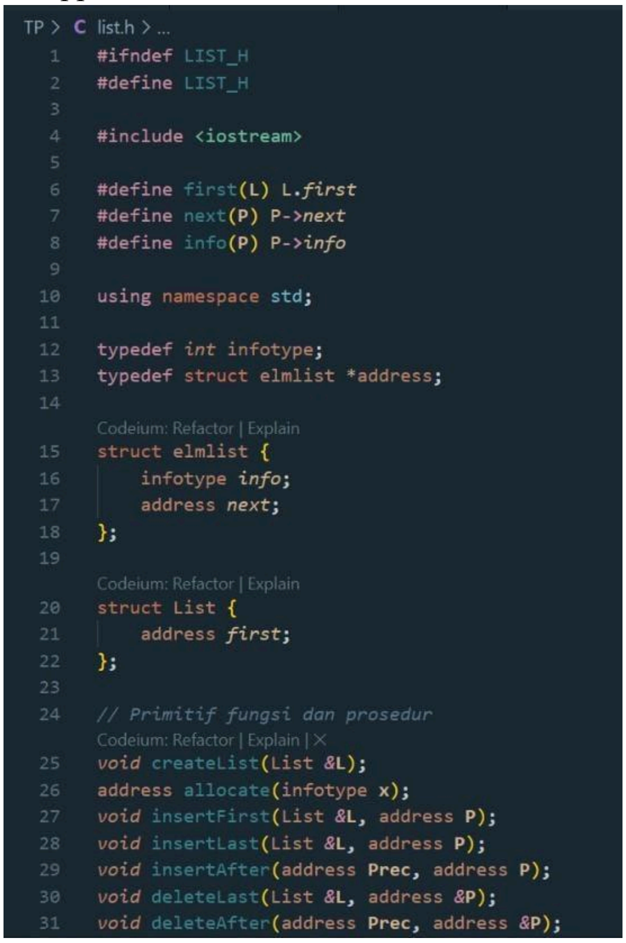


```
}

Codeium: Refactor | Explain | Generate Function Comment | ×
address searchInfo(List L, infotype x) {
    address P = first(L);
    while (P != NULL && info(P) != x) {
        P = next(P);
    }
    return P;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void printInfo(List L) {
    address P = first(L);
    while (P != NULL) {
        cout << info(P);
        P = next(P);
    }
    cout << endl;
}

```

Penjelasan:

a. lish.h
   - first(L) : mengambil elemen pertama dari list L, next(P): mengambil elemen dari node P, dan info(P): mengambil informasi dari node P
   - infotype digunakan untuk informasi yang disimpan dalam list.
   - addres digunakan untuk menunjuk ke elemen-elemen list.
   - elmlist: node linked list yang memiliki dua elemen
   - List : linked list itu sendiri yang hanya berisi satu elemen first.

list.cpp

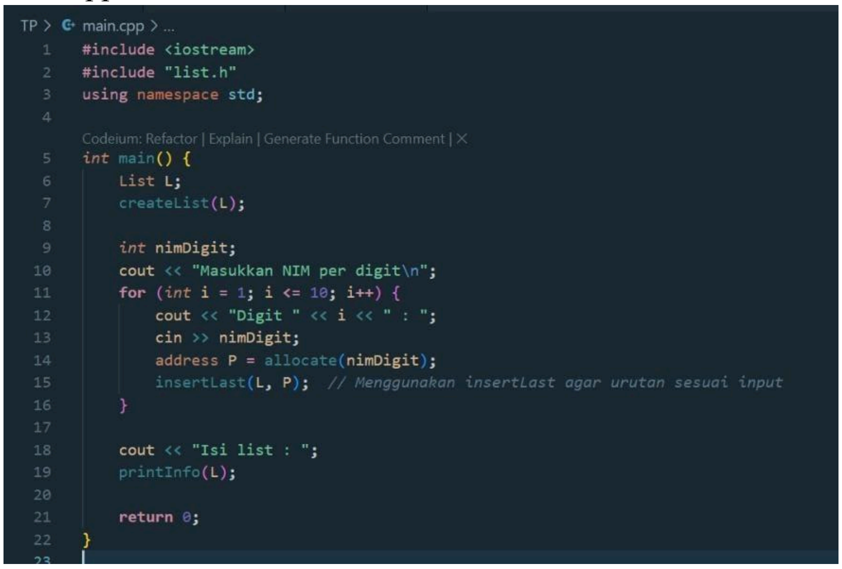


```
TP > C list.h > ...
#ifndef LIST_H
#define LIST_H

#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;

Codeium: Refactor | Explain
struct elmlist {
    infotype info;
    address next;
};

Codeium: Refactor | Explain
struct List {
    address first;
};

// Primitif fungsi dan prosedur
Codeium: Refactor | Explain | ×
void createList(List &L);
address allocate(infotype x);
void insertFirst(List &L, address P);
void insertLast(List &L, address P);
void insertAfter(address Prec, address P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
```

Penjelasan: Berisi implementasi dari fungsi-fungsi yang digunakan untuk manipulasi

Single Linked List, seperti membuat list, menambah node, menghapus node, mencari node, dan mencetak isi list.

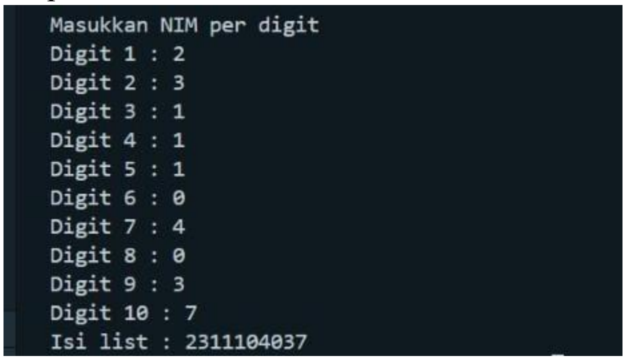
main.cpp



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

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {
    List L;
    createList(L);

    int nimDigit;
    cout << "Masukkan NIM per digit\n";
    for (int i = 1; i <= 10; i++) {
        cout << "Digit " << i << " : ";
        cin >> nimDigit;
        address P = allocate(nimDigit);
        insertLast(L, P);  // Menggunakan insertLast agar urutan sesuai input
    }

    cout << "Isi list : ";
    printInfo(L);

    return 0;
}
```

Penjelasan: fungsi utama dari program yaitu menginisialisasi list, menerima input dari pengguna (digit NIM), menambahkan digit ke dalam list, dan kemudian mencetak isi dari list.

Output:

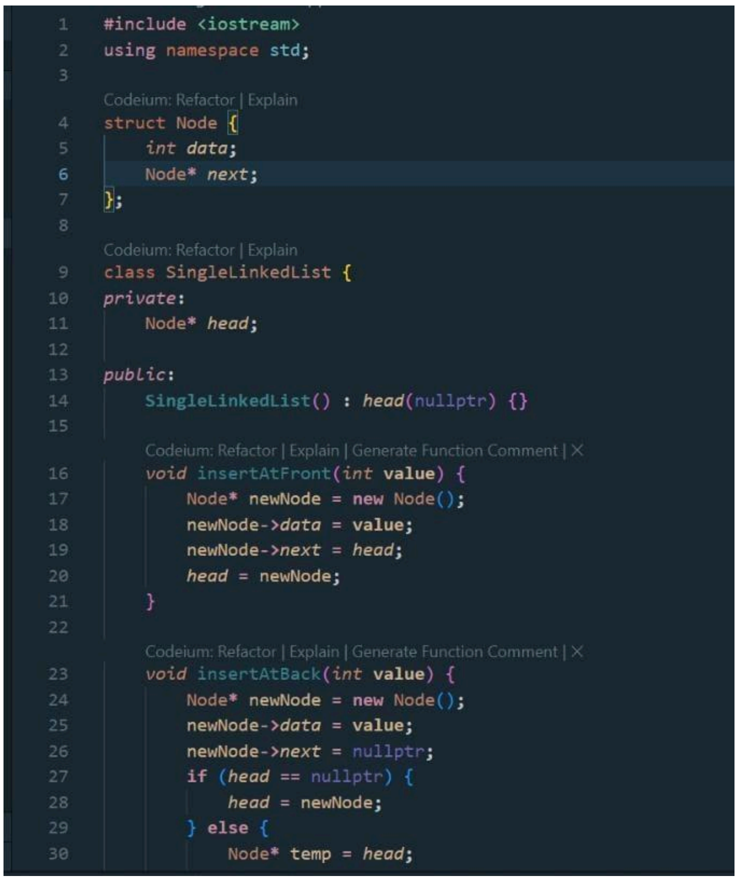


```
Masukkan NIM per digit
Digit 1 : 2
Digit 2 : 3
Digit 3 : 1
Digit 4 : 1
Digit 5 : 1
Digit 6 : 0
Digit 7 : 4
Digit 8 : 0
Digit 9 : 3
Digit 10 : 7
Isi list : 2311104037
```

Penjelasan: Ketika sudah memasukan NIM perdigit maka program akan menghasilkan output seperti diatas.

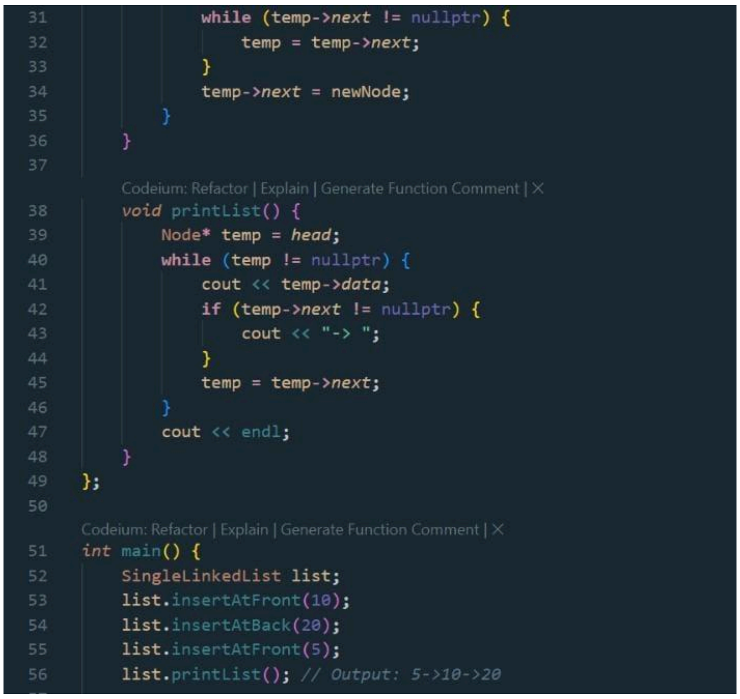
II. UNGUIDED

1. Kode:



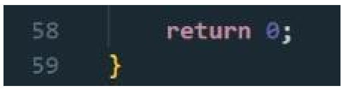
```cpp
#include <iostream>
using namespace std;

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

class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() : head(nullptr) {}

    void insertAtFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    void insertAtBack(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
```

Codeium: Refactor | Explain
Codeium: Refactor | Explain
Codeium: Refactor | Explain | Generate Function Comment | ×
Codeium: Refactor | Explain | Generate Function Comment | ×



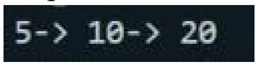
```cpp
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    void printList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << "-> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    SingleLinkedList list;
    list.insertAtFront(10);
    list.insertAtBack(20);
    list.insertAtFront(5);
    list.printList(); // Output: 5->10->20
```

Codeium: Refactor | Explain | Generate Function Comment | ×
Codeium: Refactor | Explain | Generate Function Comment | ×



```cpp
    return 0;
}
```

Output:



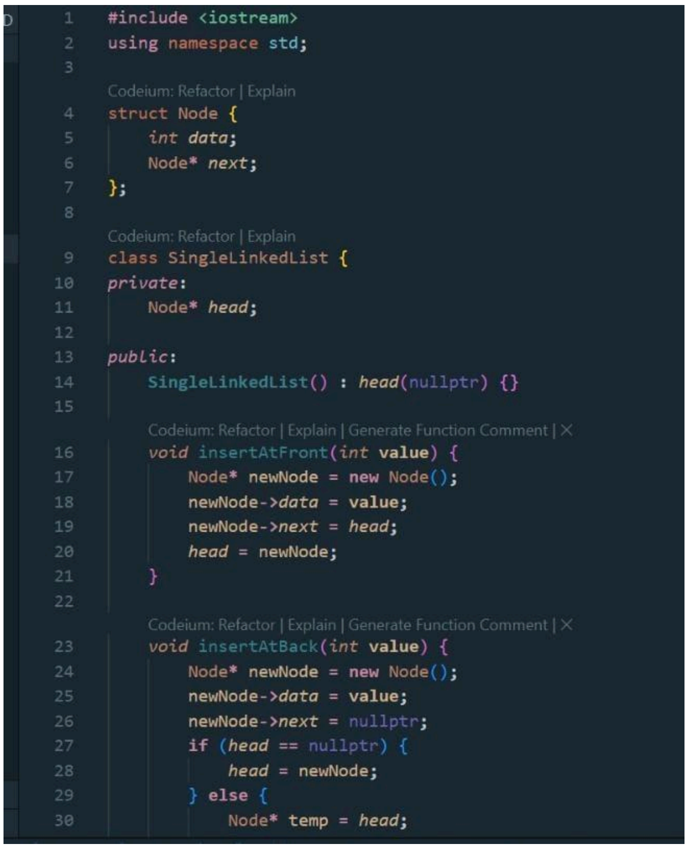
```
5-> 10-> 20
```

Penjelasan: Kode mengimplementasikan Single Linked List yang mendukung operasi dasar:

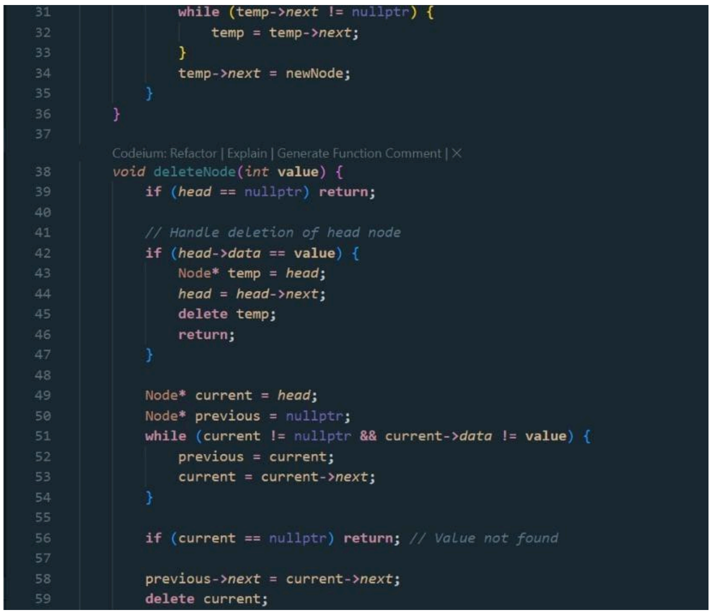
- Menambahkan node di awal (front) dan akhir (back).
- Mencetak semua elemen dalam list. Output yang dihasilkan adalah

representasi elemen-elemen dalam linked list sesuai urutan input yang diberikan.

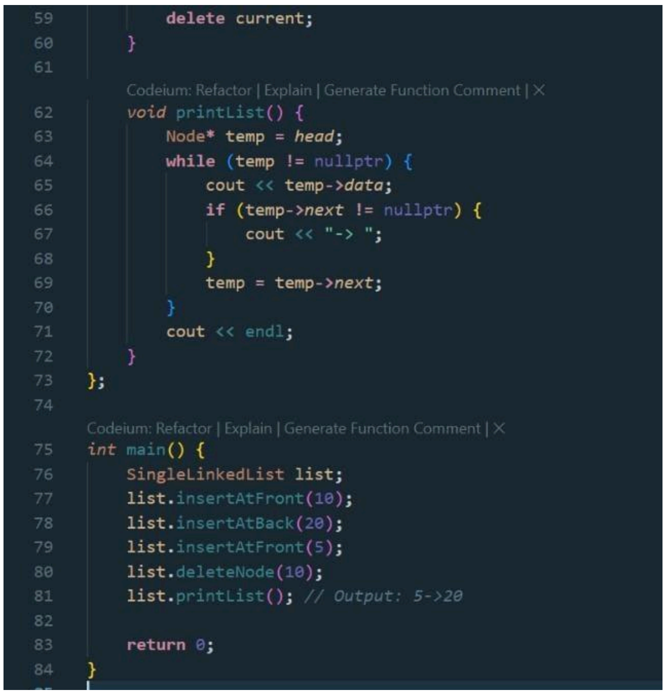
2. Kode:


Codeium: Refactor | Explain
Codeium: Refactor | Explain
Codeium: Refactor | Explain | Generate Function Comment | ×
Codeium: Refactor | Explain | Generate Function Comment | ×

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

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

class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() : head(nullptr) {}

    void insertAtFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    void insertAtBack(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
```

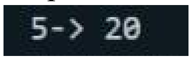


```cpp
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    void deleteNode(int value) {
        if (head == nullptr) return;

        // Handle deletion of head node
        if (head->data == value) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Node* current = head;
        Node* previous = nullptr;
        while (current != nullptr && current->data != value) {
            previous = current;
            current = current->next;
        }

        if (current == nullptr) return; // Value not found

        previous->next = current->next;
        delete current;
```



```
        delete current;
    }

    Codeium: Refactor | Explain | Generate Function Comment | X
    void printList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << "-> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
};

Codeium: Refactor | Explain | Generate Function Comment | X
int main() {
    SingleLinkedList list;
    list.insertAtFront(10);
    list.insertAtBack(20);
    list.insertAtFront(5);
    list.deleteNode(10);
    list.printList(); // Output: 5->20

    return 0;
}
```

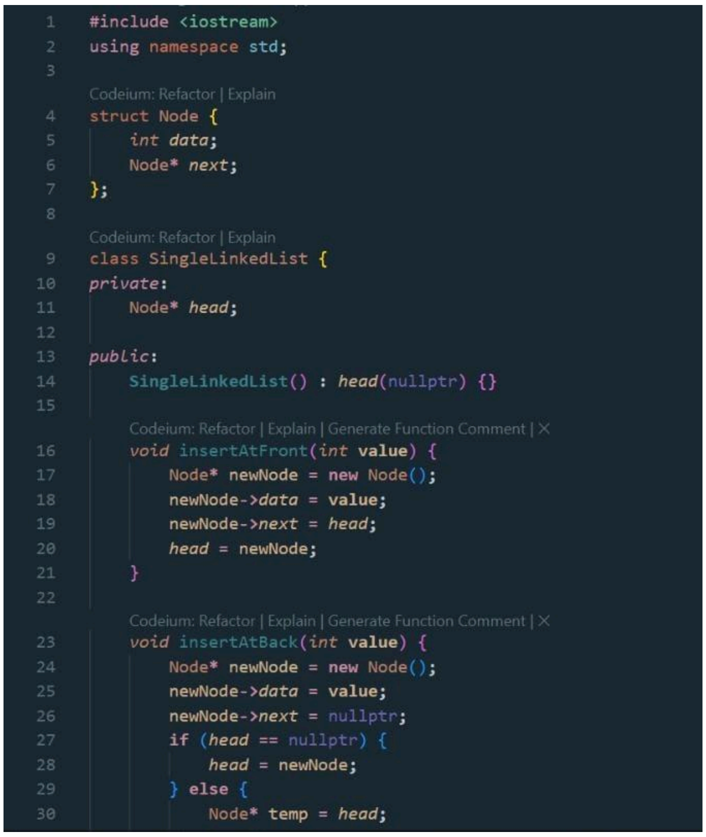
Output:



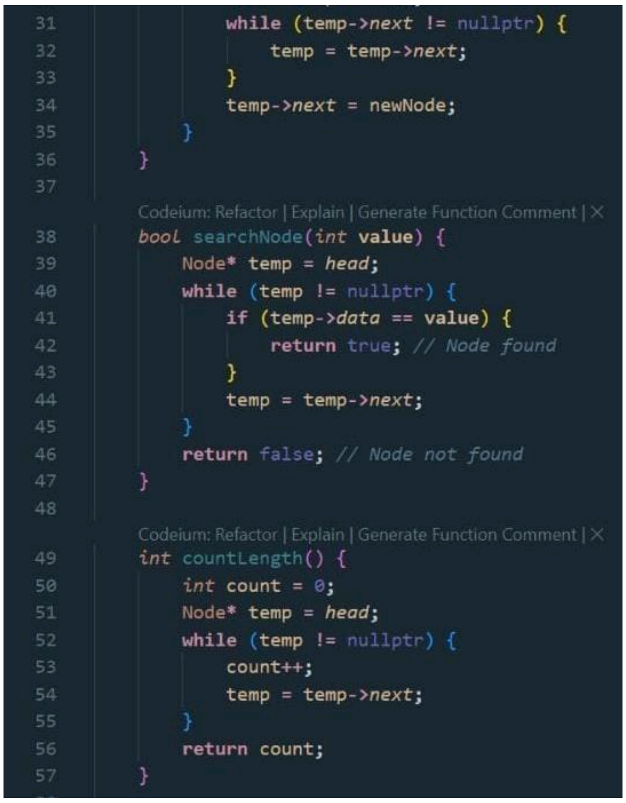
```
5-> 20
```

Penjelasan: Kode menunjukkan cara mengelola linked list dengan operasi menambah node di depan, dibelakang, menghapus node tertentu, dan mencetak elemen list. Output akhirnya adalah 5->20.

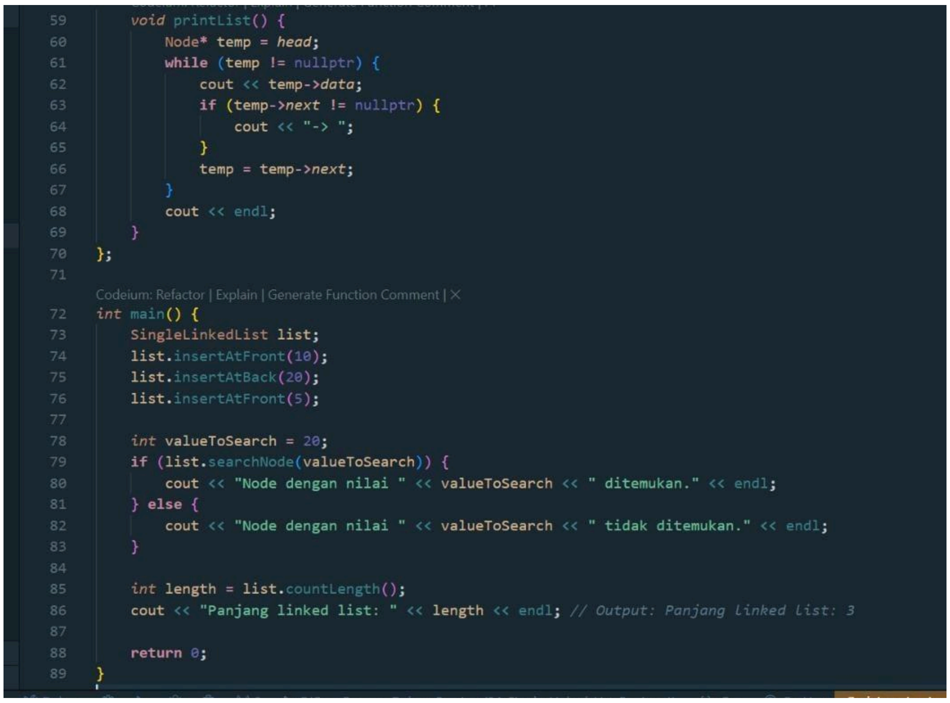
3. Kode:



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

Codeium: Refactor | Explain
struct Node {
    int data;
    Node* next;
};

Codeium: Refactor | Explain
class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() : head(nullptr) {}

    Codeium: Refactor | Explain | Generate Function Comment | X
    void insertAtFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    Codeium: Refactor | Explain | Generate Function Comment | X
    void insertAtBack(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
```

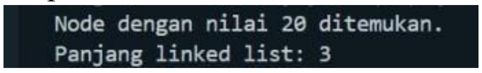

Codeium: Refactor | Explain | Generate Function Comment | ×
Codeium: Refactor | Explain | Generate Function Comment | ×

```cpp
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    bool searchNode(int value) {
        Node* temp = head;
        while (temp != nullptr) {
            if (temp->data == value) {
                return true; // Node found
            }
            temp = temp->next;
        }
        return false; // Node not found
    }

    int countLength() {
        int count = 0;
        Node* temp = head;
        while (temp != nullptr) {
            count++;
            temp = temp->next;
        }
        return count;
    }
```



```cpp
    void printList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << "-> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    SingleLinkedList list;
    list.insertAtFront(10);
    list.insertAtBack(20);
    list.insertAtFront(5);

    int valueToSearch = 20;
    if (list.searchNode(valueToSearch)) {
        cout << "Node dengan nilai " << valueToSearch << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << valueToSearch << " tidak ditemukan." << endl;
    }

    int length = list.countLength();
    cout << "Panjang linked list: " << length << endl; // Output: Panjang linked list: 3

    return 0;
}
```

Output:



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

Penjelasan: Kode mendemonstrasikan operasi dasar pada single linked list, dengan menambah elemen di depan dan belakang, mencari elemen, menghitung panjang, dan mencetak isi list. Outputnya adalah informasi apakah nilai yang dicari ditemukan dan panjang list (3).