

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 3
SINGLE AND
DOUBLE LINKED LIST**



Disusun Oleh :
Muhammad Hamzah Haifan Ma'ruf
231102091

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

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

A. Dasar Teori

Linked list adalah struktur data linier berbentuk rantai simpul di mana setiap simpul menyimpan 2 item, yaitu nilai data dan pointer ke simpul elemen berikutnya. Berbeda dengan array, elemen linked list tidak ditempatkan dalam alamat memori yang berdekatan melainkan elemen ditautkan menggunakan pointer. Simpul pertama dari linked list disebut sebagai head atau simpul kepala. Apabila linked list berisi elemen kosong, maka nilai pointer dari head menunjuk ke NULL. Begitu juga untuk pointer berikutnya dari simpul terakhir atau simpul ekor akan menunjuk ke NULL. Ukuran elemen dari linked list dapat bertambah secara dinamis dan mudah untuk menyisipkan dan menghapus elemen karena tidak seperti array, kita hanya perlu mengubah pointer elemen sebelumnya dan elemen berikutnya untuk menyisipkan atau menghapus elemen.

Jenis Jenis Linked List

1. Singly linked list

Singly linked list adalah linked list unidirectional. Jadi, kita hanya dapat melintasinya dalam satu arah, yaitu dari simpul kepala ke simpul ekor.

2. Doubly linked list

Doubly linked list adalah linked list bidirectional. Jadi, kita bisa melintasinya secara dua arah. Tidak seperti singly linked list, simpul doubly linked list berisi satu pointer tambahan yang disebut previous pointer. Pointer ini menunjuk ke simpul sebelumnya.

3. Circular linked list

Circular linked list adalah linked list unidirectional. Kita hanya dapat melintasinya dalam satu arah. Tetapi jenis linked list ini memiliki simpul terakhir yang menunjuk ke simpul kepala. Jadi saat melintas, kita harus berhati-hati dan berhenti saat mengunjungi kembali simpul kepala.

4. Circular doubly linked list

Circular doubly linked list adalah gabungan dari Doubly linked list dan Circular linked list. Seperti Doubly linked list, linked list ini memiliki pointer tambahan yang disebut previous pointer, dan mirip dengan Circular linked list, simpul terakhirnya menunjuk pada simpul kepala. Jenis linked list ini adalah bidirectional. Jadi, kita bisa melintasinya dua arah.

Karakteristik Linked List

- Linked list menggunakan memori tambahan untuk menyimpan link (tautan)
- Untuk inisialisasi awal linked list, kita tidak perlu tahu ukuran dari elemen.
- Linked list umumnya dapat digunakan untuk mengimplementasikan struktur data lain seperti stack, queue, ataupun graf
- Simpul pertama dari linked list disebut sebagai Head.
- Pointer setelah simpul terakhir selalu bernilai NULL
- Dalam struktur data linked list, operasi penyisipan dan penghapusan dapat dilakukan dengan mudah
- Tiap-tiap simpul dari linked list berisi pointer atau tautan yang menjadi alamat dari simpul berikutnya

- Linked list bisa menyusut atau bertambah kapan saja dengan mudah.

Operasi-operasi pada Linked List

- Traversal - mengakses setiap elemen dari linked list
- Insertion - menambahkan elemen baru ke linked list
- Deletion - menghapus elemen yang ada
- Searching - menemukan simpul pada linked list
- Sorting - mengurutkan simpul dari struktur linked list

Fungsi dan Kegunaan Linked List

- Digunakan untuk melakukan operasi aritmatika pada bilangan long integer
- Dipakai untuk representasi matriks rongga.
- Digunakan dalam alokasi file yang ditautkan.
- Membantu dalam manajemen memori.

Guided 1

Source Code

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

//PROGRAM SINGLE LINKED LIST NON-CIRCULAR

//Deklarasi Struct Node
struct Node{
    int data;
    Node *next;
};

Node *head;
Node *tail;

//Inisialisasi Node
void init(){
    head = NULL;
    tail = NULL;
}

// Pengecekan
bool isEmpty(){
    if (head == NULL)
        return true;
    else
        return false;
}

//Tambah Depan
void insertDepan(int nilai){
    //Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty() == true){
        head = tail = baru;
        tail->next = NULL;
    }
    else{
        baru->next = head;
        head = baru;
    }
}

//Tambah Belakang
void insertBelakang(int nilai){
    //Buat Node baru
```

```

    Node *baru = new Node;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty() == true){
        head = tail = baru;
        tail->next = NULL;
    }
    else{
        tail->next = baru;
        tail = baru;
    }
}

//Hitung Jumlah List
int hitungList(){
    Node *hitung;
    hitung = head;
    int jumlah = 0;
    while( hitung != NULL ){
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}

//Tambah Tengah
void insertTengah(int data, int posisi){
    if( posisi < 1 || posisi > hitungList() ){
        cout << "Posisi diluar jangkauan" << endl;
    }
    else if( posisi == 1){
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else{
        Node *baru, *bantu;
        baru = new Node();
        baru->data = data;
        // tranversing
        bantu = head;
        int nomor = 1;
        while( nomor < posisi - 1 ){
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

```

```

//Hapus Depan
void hapusDepan() {
    Node *hapus;
    if (isEmpty() == false){
        if (head->next != NULL){
            hapus = head;
            head = head->next;
            delete hapus;
        }
        else{
            head = tail = NULL;
        }
    }
    else{
        cout << "List kosong!" << endl;
    }
}

//Hapus Belakang
void hapusBelakang() {
    Node *hapus;
    Node *bantu;
    if (isEmpty() == false){
        if (head != tail){
            hapus = tail;
            bantu = head;
            while (bantu->next != tail){
                bantu = bantu->next;
            }
            tail = bantu;
            tail->next = NULL;
            delete hapus;
        }
        else{
            head = tail = NULL;
        }
    }
    else{
        cout << "List kosong!" << endl;
    }
}

//Hapus Tengah
void hapusTengah(int posisi){
    Node *hapus, *bantu, *bantu2;
    if( posisi < 1 || posisi > hitungList() ){
        cout << "Posisi di luar jangkauan" << endl;
    }
    else if( posisi == 1){

```

```

        cout << "Posisi bukan posisi tengah" << endl;
    }
    else{
        int nomor = 1;
        bantu = head;
        while( nomor <= posisi ){
            if( nomor == posisi-1 ){
                bantu2 = bantu;
            }
            if( nomor == posisi ){
                hapus = bantu;
            }
            bantu = bantu->next;
            nomor++;
        }
        bantu2->next = bantu;
        delete hapus;
    }
}

//Ubah Depan
void ubahDepan(int data){
    if (isEmpty() == false){
        head->data = data;
    }
    else{
        cout << "List masih kosong!" << endl;
    }
}

//Ubah Tengah
void ubahTengah(int data, int posisi){
    Node *bantu;
    if (isEmpty() == false){
        if( posisi < 1 || posisi > hitungList() ){
            cout << "Posisi di luar jangkauan" << endl;
        }
        else if( posisi == 1){
            cout << "Posisi bukan posisi tengah" << endl;
        }
        else{
            bantu = head;
            int nomor = 1;
            while (nomor < posisi){
                bantu = bantu->next; nomor++;
            }
            bantu->data = data;
        }
    }
}

```

```

        else{
            cout << "List masih kosong!" << endl;
        }
    }

    //Ubah Belakang
    void ubahBelakang(int data){
        if (isEmpty() == false){
            tail->data = data;
        }
        else{
            cout << "List masih kosong!" << endl;
        }
    }

    //Hapus List
    void clearList(){
        Node *bantu, *hapus;
        bantu = head;
        while (bantu != NULL){
            hapus = bantu; bantu = bantu->next;
            delete hapus;
        }
        head = tail = NULL;
        cout << "List berhasil terhapus!" << endl;
    }

    //Tampilkan List
    void tampil(){
        Node *bantu;
        bantu = head;
        if (isEmpty() == false){
            while (bantu != NULL){
                cout << bantu->data << ends;
                bantu = bantu->next;
            }
            cout << endl;
        }
        else{
            cout << "List masih kosong!" << endl;
        }
    }

    int main(){
        init();
        insertDepan(3);tampil();
        insertBelakang(5);
        tampil();
        insertDepan(2);
        tampil();
        insertDepan(1);
        tampil();
    }

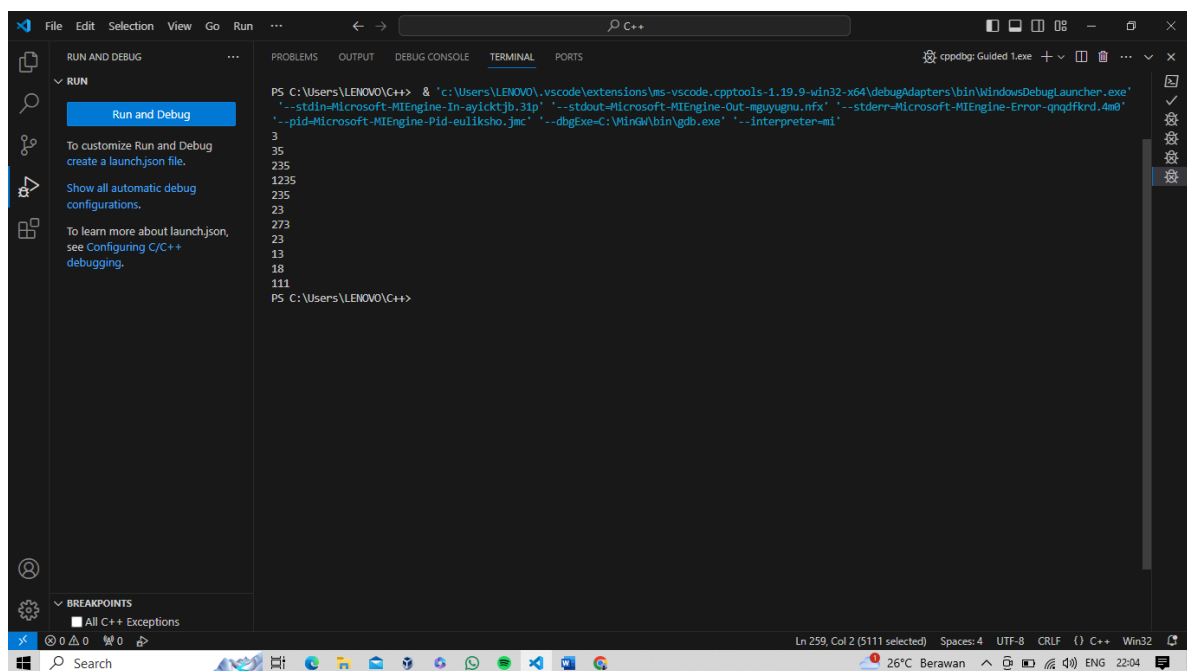
```

```

hapusDepan();
tampil();
hapusBelakang();
tampil();
insertTengah(7,2);
tampil();
hapusTengah(2);
tampil();
ubahDepan(1);
tampil();
ubahBelakang(8);
tampil();
ubahTengah(11, 2);
tampil();
return 0;
}

```

Screenshots Output :



```

PS C:\Users\LENOVO\C++ > & 'c:\Users\LENOVO\.vscode\extensions\ms-vscode.cpptools-1.19.9-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe'
'--stdin=Microsoft-MIEngine-In-ayicktjb.3ip' '--stdout=Microsoft-MIEngine-Out-nguyugnu.nfx' '--stderr=Microsoft-MIEngine-Error-qndfkrnd.4m'
'--pid=Microsoft-MIEngine-Pid-euliksxo.jmc' '--dbgExe=C:\Windows\bin\gdb.exe' '--interpreter=mi'
3
35
235
1235
235
23
273
23
13
18
111
PS C:\Users\LENOVO\C++ >

```

Deskripsi :

Program ini adalah implementasi dari single linked list non-circular menggunakan bahasa pemrograman C++. Program ini menyediakan fungsi-fungsi untuk melakukan operasi-operasi dasar pada linked list, seperti menambahkan node di depan, di belakang, di tengah, menghapus node di depan, di belakang, di tengah, serta mengubah nilai data pada node tertentu. Selain itu, program juga memiliki fungsi untuk menghitung jumlah node dalam linked list serta membersihkan seluruh isi linked list. Setiap operasi pada linked list dapat ditampilkan hasilnya melalui fungsi tampil(). Program diawali dengan inisialisasi linked list, dilanjutkan dengan

pemanggilan serangkaian fungsi untuk memanipulasi linked list, seperti menambah, menghapus, dan mengubah isi dari node-node yang ada.

Guided 2

Source code

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }
    void push(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;
        if (head != nullptr) {
            head->prev = newNode;
        }
        else {
            tail = newNode;
        }
        head = newNode;
    }
    void pop() {
        if (head == nullptr) {
            return;
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
        else {
            tail = nullptr;
        }
        delete temp;
    }
};
```

```

    }
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true;
            }
            current = current->next;
        }
        return false;
    }
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;
        int choice;
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1: {
                int data;
                cout << "Enter data to add: ";
                cin >> data;
            }
        }
    }
}

```

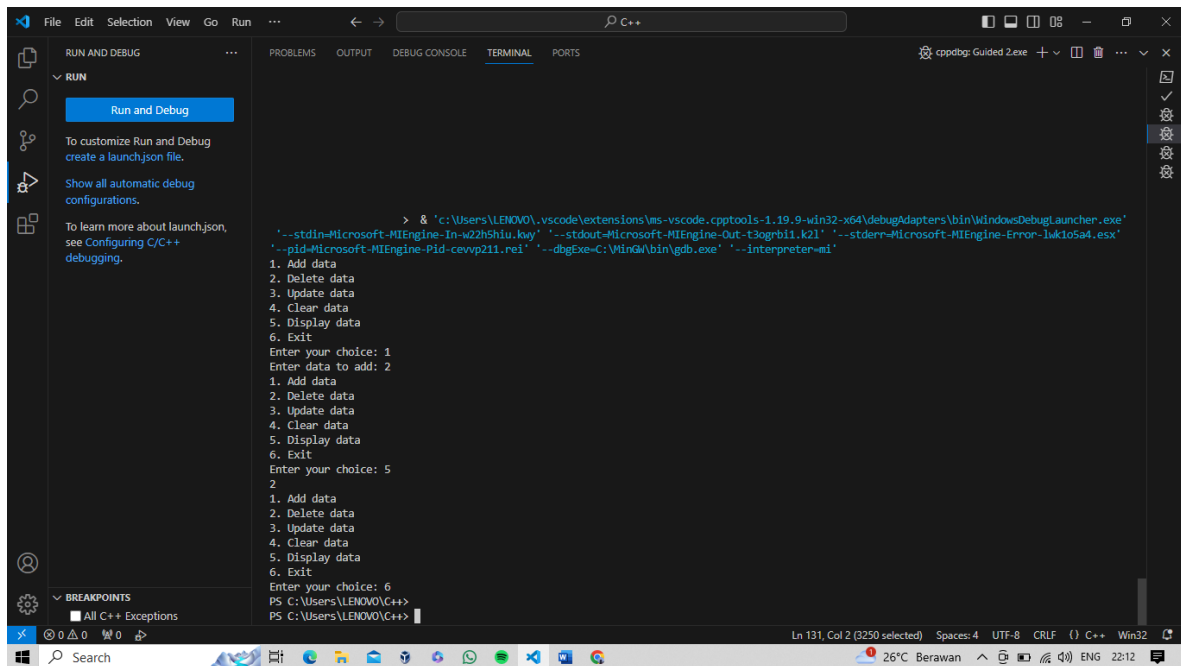
```

        list.push(data);
        break;
    }
    case 2: {
        list.pop();
        break;
    }
    case 3: {
        int oldData, newData;
        cout << "Enter old data: ";
        cin >> oldData;
        cout << "Enter new data: ";
        cin >> newData;
        bool updated = list.update(oldData, newData);
        if (!updated) {
            cout << "Data not found" << endl;
        }
        break;
    }
    case 4: {
        list.deleteAll();
        break;
    }
    case 5: {
        list.display();
        break;
    }
    case 6: {
        return 0;
    }
    default: {
        cout << "Invalid choice" << endl;
        break;
    }
}

return 0;
}

```

Screenshots Output :



```
> & 'c:\Users\LENOVO\.vscode\extensions\ms-vscode.cpptools-1.19.9-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe'
'--stdin-Microsoft-MIEngine-In-w02h5hlu.kuy' '--stdout-Microsoft-MIEngine-Out-t3ogrbi1.k21' '--stderr-Microsoft-MIEngine-Error-lwk105a4.esx'
'--pid-Microsoft-MIEngine-Pid-cevvp211.rei' '--dbgExe=C:\MinGW\bin\gdb.exe' '--interpreter=mi'
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
PS C:\Users\LENOVO\C++>
PS C:\Users\LENOVO\C++>
```

Deskripsi :

Program di atas adalah implementasi dari Doubly Linked List menggunakan bahasa pemrograman C++. Program ini mengimplementasikan kelas Node yang memiliki tiga atribut yaitu data, pointer prev, dan pointer next. Kelas DoublyLinkedList memiliki atribut head dan tail yang merepresentasikan pointer ke node pertama dan terakhir dalam linked list. Program menyediakan fungsi-fungsi seperti push untuk menambahkan node di depan, pop untuk menghapus node di depan, update untuk mengubah nilai data pada node tertentu, deleteAll untuk menghapus seluruh isi linked list, dan display untuk menampilkan isi linked list. Program diawali dengan loop tak terbatas yang memberikan opsi kepada pengguna untuk menambah, menghapus, mengubah, menghapus semua data, menampilkan data, atau keluar dari program. Sesuai dengan pilihan pengguna, program akan melakukan operasi yang sesuai pada Doubly Linked List tersebut.

C. Unguided

Unguided 1

Source code

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

// Deklarasi Struct Node
struct Node {
    string Nama_091;
    int Umur;
    Node* next;
};

Node* head;
Node* tail;

// Inisialisasi Node
void inisialisasi() {
    head = NULL;
    tail = NULL;
}

// Pengecekan
bool cek() {
    if (head == NULL)
        return true;
    else
        return false;
}

// Tambah Depan
void depan(string name, int age) {
    //Buat Node baru
    Node* baru = new Node;
    baru->Nama_091 = name;
    baru->Umur = age;
    baru->next = NULL;

    if (cek() == true) {
        head = tail = baru;
        tail->next = NULL;
    }
    else {
        baru->next = head;
        head = baru;
    }
}
```

```

    }
}

// Tambah Belakang
void belakang(string name, int age) {
    // Buat Node baru
    Node* baru = new Node;
    baru->Nama_091 = name;
    baru->Umur = age;
    baru->next = NULL;

    if (cek() == true) {
        head = tail = baru;
        tail->next = NULL;
    }
    else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int jumlahlist() {
    Node* hitung;
    hitung = head;
    int jumlah = 0;

    while (hitung != NULL) {
        jumlah++;
        hitung = hitung->next;
    }

    return jumlah;
}

// Tambah Tengah
void tengah(string name, int age, int posisi) {
    if (posisi < 1 || posisi > jumlahlist()) {
        cout << "Tidak terjangkau!" << endl;
    }
    else if (posisi == 1) {
        cout << "Bukan di tengah." << endl;
    }
    else {
        Node* baru, * bantu;
        baru = new Node();
        baru->Nama_091 = name;
        baru->Umur = age;
    }
}

```

```

        // Tranversing
        bantu = head;
        int nomor = 1;

        while (nomor < posisi - 1) {
            bantu = bantu->next;
            nomor++;
        }

        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Depan
void hapus() {
    Node* hapus;

    if (cek() == false) {
        if (head->next != NULL) {
            hapus = head;
            head = head->next;
            delete hapus;
        }
        else {
            head = tail = NULL;
        }
    }
    else {
        cout << "Kosong!" << endl;
    }
}

// Hapus Belakang
void hapusbelakang() {
    Node* hapus;
    Node* bantu;

    if (cek() == false) {
        if (head != tail) {
            hapus = tail;
            bantu = head;

            while (bantu->next != tail) {
                bantu = bantu->next;
            }

            tail = bantu;
            tail->next = NULL;
        }
    }
}

```

```

        delete hapus;
    }
    else {
        head = tail = NULL;
    }
}
else {
    cout << "Kosong!" << endl;
}
}

// Hapus Tengah
void hapustengah(int posisi) {
    Node* hapus, * bantu, * bantu2;

    if (posisi < 1 || posisi > jumlahlist()) {
        cout << "Tidak terjangkau!" << endl;
    }
    else if (posisi == 1) {
        cout << "Bukan yang tengah." << endl;
    }
    else {
        int nomor = 1;
        bantu = head;

        while (nomor <= posisi) {
            if (nomor == posisi - 1) {
                bantu2 = bantu;
            }

            if (nomor == posisi) {
                hapus = bantu;
            }

            bantu = bantu->next;
            nomor++;
        }

        bantu2->next = bantu;
        delete hapus;
    }
}

// Ubah Depan
void ubahdepan(string name, int age) {
    if (cek() == false) {
        head->Nama_091 = name;
        head->Umur = age;
    }
}

```

```

        else {
            cout << "Tidak ada yang berubah!" << endl;
        }
    }

// Ubah Tengah
void ubahTengah(string name, int age, int posisi) {
    Node* bantu;

    if (cek() == false) {
        if (posisi < 1 || posisi > jumlahlist()) {
            cout << "Tidak Terjangkau!" << endl;
        }
        else if (posisi == 1) {
            cout << "Bukan yang Tengah." << endl;
        }
        else {
            bantu = head;
            int nomor = 1;

            while (nomor < posisi) {
                bantu = bantu->next;
                nomor++;
            }

            bantu->Nama_091 = name;
            bantu->Umur = age;
        }
    }
    else {
        cout << "Kosong!" << endl;
    }
}

// Ubah Belakang
void ubahbelakang(string name, int age) {
    if (cek() == false) {
        tail->Nama_091 = name;
        tail->Umur = age;
    }
    else {
        cout << "Kosong" << endl;
    }
}

// Hapus List
void hapuslist() {
    Node* bantu, * hapus;
    bantu = head;

```

```

        while (bantu != NULL) {
            hapus = bantu;
            bantu = bantu->next;
            delete hapus;
        }

        head = tail = NULL;
        cout << "Menghapus semua!" << endl;
    }

// Tampilkan List
void tampilkanlist() {
    Node* bantu;
    bantu = head;

    cout << left << setw(15) << "Nama" << right << setw(4) << "Usia" <<
endl;

    if (cek() == false) {
        while (bantu != NULL) {
            cout << left << setw(15) << bantu->Nama_091 << right <<
setw(4) << bantu->Umur << endl;
            bantu = bantu->next;
        }

        cout << endl;
    }
    else {
        cout << "Kosong!" << endl;
    }
}

int main() {
    // Inisialisasi Linked List
    inisialisasi();

    // Menampilkan nama dan umur awal & menjawab poin a
    cout << "(A) Menampilkan data sesuai urutan data pertama nama dan
usia user\n" << endl;
    depan("Karin", 18);
    depan("Hoshino", 18);
    depan("Akechi", 20);
    depan("Yusuke", 19);
    depan("Michael", 18);
    depan("Jane", 20);
    depan("John", 19);
    depan("Ipin", 19);
    tampilkanlist();
}

```

```

// b. Menghapus data Akheci
cout << "(B) Hapus data Akechi\n" << endl;
hapustengah(6);
tampilist();

// c. Menambahkan data Futaba (18) diantara John dan Jane
cout << "(C) Tambah data Futaba (18) diantara John & Jane\n" <<
endl;
tengah("Futaba", 18, 3);
tampilist();

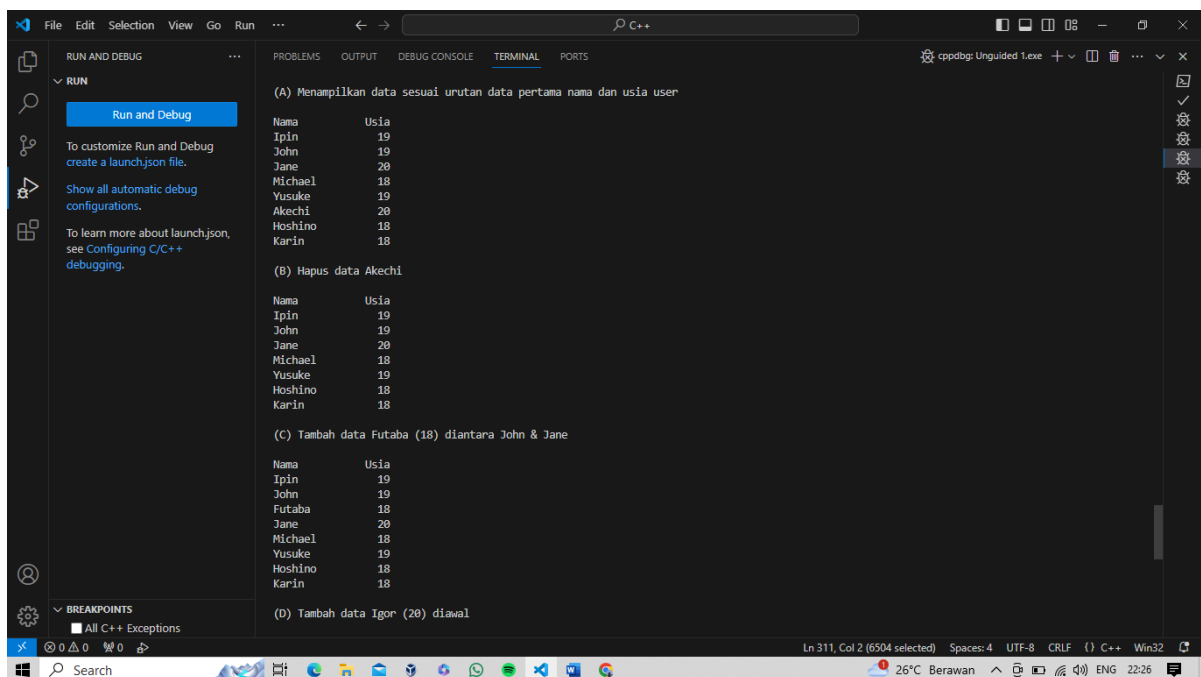
// d. Menambahkan data Igor diawal (20)
cout << "(D) Tambah data Igor (20) diawal\n" << endl;
depan("Igor", 20);
tampilist();

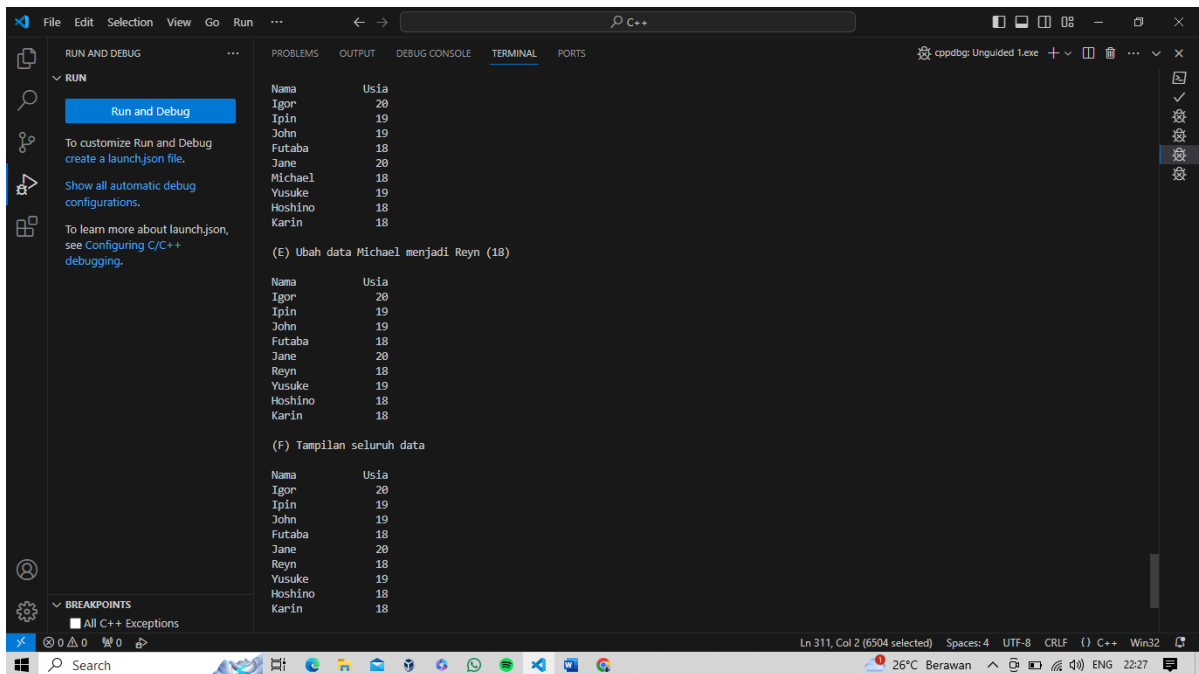
// e. Mengubah data Michael menjadi Reyn (18)
cout << "(E) Ubah data Michael menjadi Reyn (18)\n" << endl;
ubahtengah("Reyn", 18, 6);
tampilist();

// f. Menampillkan seuruh data
cout << "(F) Tampilan seluruh data\n" << endl;
tampilist();
return 0;
}

```

Screenshots Output :





Deskripsi :

Program di atas adalah program menggunakan struktur data Node yang menyimpan informasi tentang nama (string) dan umur (integer) dari seseorang. Kemudian, program menyediakan fungsi-fungsi untuk melakukan operasi-operasi seperti menambahkan node di depan, di belakang, di tengah, menghapus node di depan, di belakang, di tengah, mengubah nilai data pada node tertentu, dan menampilkan seluruh isi linked list. Program diawali dengan inisialisasi linked list, kemudian dilanjutkan dengan serangkaian operasi sesuai dengan pilihan yang ditentukan di dalam fungsi main(). Setiap operasi yang dilakukan akan ditampilkan hasilnya melalui fungsi tampillist().

Unguided 2

Source Code

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

// Deklarasi Class Node untuk Double LL
class Node {
public:
    string Nama_Produk_091;
    int harga;
    Node* prev;
    Node* next;
};

// Deklarasi Class DoublyLL untuk Double LL
class DoublyLinkedList {
public:
    Node* head;
    Node* tail;
```

```

DoublyLinkedList() {
    head = nullptr;
    tail = nullptr;
}

// Menambahkan produk ke dalam LL di bagian atas
void tambahproduk(string Nama_Produk_091, int harga) {
    Node* newNode = new Node;
    newNode->Nama_Produk_091 = Nama_Produk_091;
    newNode->harga = harga;
    newNode->prev = nullptr;
    newNode->next = head;
    if (head != nullptr) {
        head->prev = newNode;
    }
    else {
        tail = newNode;
    }
    head = newNode;
}

// Menghapus produk teratas dari LL
void hapusproduk() {
    if (head == nullptr) {
        return;
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    }
    else {
        tail = nullptr;
    }
    delete temp;
}

// Mengubah data produk berdasarkan nama produk
bool ubahproduk(string Nama_Produk_Lama, string Nama_Produk_Baru, int
Harga_Baru) {
    Node* current = head;
    while (current != nullptr) {
        if (current->Nama_Produk_091 == Nama_Produk_Lama) {
            current->Nama_Produk_091 = Nama_Produk_Baru;
            current->harga = Harga_Baru;
            return true;
        }
        current = current->next;
    }
}

```

```

        return false;
    }

    // Menambahkan data produk pada posisi tertentu
    void sisipposisi(string Nama_Produk_091, int harga, int posisi) {
        if (posisi < 1) {
            cout << "Posisi tidak ada" << endl;
            return;
        }
        // Jika posisi adalah 1 maka tambahkan data produk di depan
linked list
        Node* newNode = new Node;
        newNode->Nama_Produk_091 = Nama_Produk_091;
        newNode->harga = harga;
        if (posisi == 1) {
            newNode->next = head;
            newNode->prev = nullptr;
            if (head != nullptr) {
                head->prev = newNode;
            }
            else {
                tail = newNode;
            }
            head = newNode;
            return;
        }
        // Looping sampai posisi sebelum posisi yang diinginkan
        Node* current = head;
        for (int i = 1; i < posisi - 1 && current != nullptr; ++i) {
            current = current->next;
        }
        if (current == nullptr) {
            cout << "Posisi tidak ada" << endl;
            return;
        }
        newNode->next = current->next;
        newNode->prev = current;
        // Pointer prev node setelah current menunjuk ke newNode jika
node setelah current tidak nullptr
        if (current->next != nullptr) {
            current->next->prev = newNode;
        }
        else {
            tail = newNode;
        }
        current->next = newNode;
    }

    // Menghapus data produk pada posisi tertentu

```

```

void hapusposisi(int posisi) {
    if (posisi < 1 || head == nullptr) {
        cout << "Posisi tidak ada atau list kosong" << endl;
        return;
    }
    Node* current = head;
    if (posisi == 1) {
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
        else {
            tail = nullptr;
        }
        delete current;
        return;
    }
    // Looping sampai posisi yang diinginkan
    for (int i = 1; current != nullptr && i < posisi; ++i) {
        current = current->next;
    }
    if (current == nullptr) {
        cout << "Posisi tidak ada" << endl;
        return;
    }
    if (current->next != nullptr) {
        current->next->prev = current->prev;
    }
    else {
        tail = current->prev;
    }
    current->prev->next = current->next;
    delete current;
}

// Menghapus semua data produk
void hapussemua() {
    Node* current = head;
    while (current != nullptr) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    head = nullptr;
    tail = nullptr;
}

// Menampilkan data produk
void tampilan() {

```

```

        Node* current = head;
        cout << "\nDaftar Produk dan harga yang tersedia :" << endl;
        cout << left << setw(20) << "Nama Produk" << "Harga" << endl;
        while (current != nullptr) {
            cout << left << setw(20) << current->Nama_Produk_091 <<
current->harga << endl;
            current = current->next;
        }
        cout << endl;
    }
};

// Deklarasi objek list dari class DoublyLL
int main() {
    DoublyLinkedList list;

    list.tambahproduk("Hanasui", 30000);
    list.tambahproduk("Wardah", 50000);
    list.tambahproduk("Skintific", 100000);
    list.tambahproduk("Somethinc", 150000);
    list.tambahproduk("Originote", 60000);

    cout << "\nToko Skincare Purwokerto" << endl;
    list.tampilan();

    // Looping menu utama
    while (true) {
        cout << "\nToko Skincare Purwokerto" << endl;
        cout << "1. Tambah Data" << endl;
        cout << "2. Hapus Data" << endl;
        cout << "3. Update Data" << endl;
        cout << "4. Tambah Data Urutan Tertentu" << endl;
        cout << "5. Hapus Data Urutan Tertentu" << endl;
        cout << "6. Hapus Seluruh Data" << endl;
        cout << "7. Tampilkan Data" << endl;
        cout << "8. Exit" << endl;
        int pilihan;
        cout << "Pilih Menu: ";
        cin >> pilihan;
        // Switch case untuk memilih menu
        switch (pilihan) {
            // Memanggil fungsi tambah_produk
            case 1: {
                string Nama_Produk_091;
                int harga;
                cout << "Masukkan nama produk: ";
                cin >> Nama_Produk_091;
                cout << "Masukkan harga: ";
                cin >> harga;
            }
        }
    }
}

```

```

        list.tambahproduk>Nama_Produk_091, harga);
        cout << "Produk berhasil ditambahkan teratas" << endl;
        break;
    }
    // Memanggil fungsi hapus_produk
    case 2: {
        list.hapusproduk();
        cout << "Produk teratas berhasil dihapus" << endl;
        break;
    }
    // Memanggil fungsi ubah_produk
    case 3: {
        string>Nama_Produk_Lama,>Nama_Produk_Baru;
        int>Harga_Baru;
        cout << "Input nama produk lama: ";
        cin >>>Nama_Produk_Lama;
        cout << "Input nama produk baru: ";
        cin >>>Nama_Produk_Baru;
        cout << "Input harga baru: ";
        cin >>>Harga_Baru;
        bool>updated = list.ubahproduk>Nama_Produk_Lama,
Nama_Produk_Baru, Harga_Baru);
        if (!updated) {
            cout << "Data produk tidak ditemukan" << endl;
        }
        else {
            cout << "Data produk berhasil diupdate" << endl;
        }
        break;
    }
    // Memanggil fungsi sisipkan_posisi_tertentu
    case 4: {
        string>Nama_Produk_091;
        int>harga,>position;
        cout << "Input nama produk: ";
        cin >>>Nama_Produk_091;
        cout << "Input harga: ";
        cin >>>harga;
        cout << "Input posisi: ";
        cin >>>position;
        list.sisipposisi>Nama_Produk_091, harga, position);
        cout << "Produk berhasil ditambahkan pada posisi " <<
position << endl;
        break;
    }
    // Memanggil fungsi hapus_posisi_tertentu
    case 5: {
        int>position;
        cout << "Input posisi yang ingin dihapus: ";

```

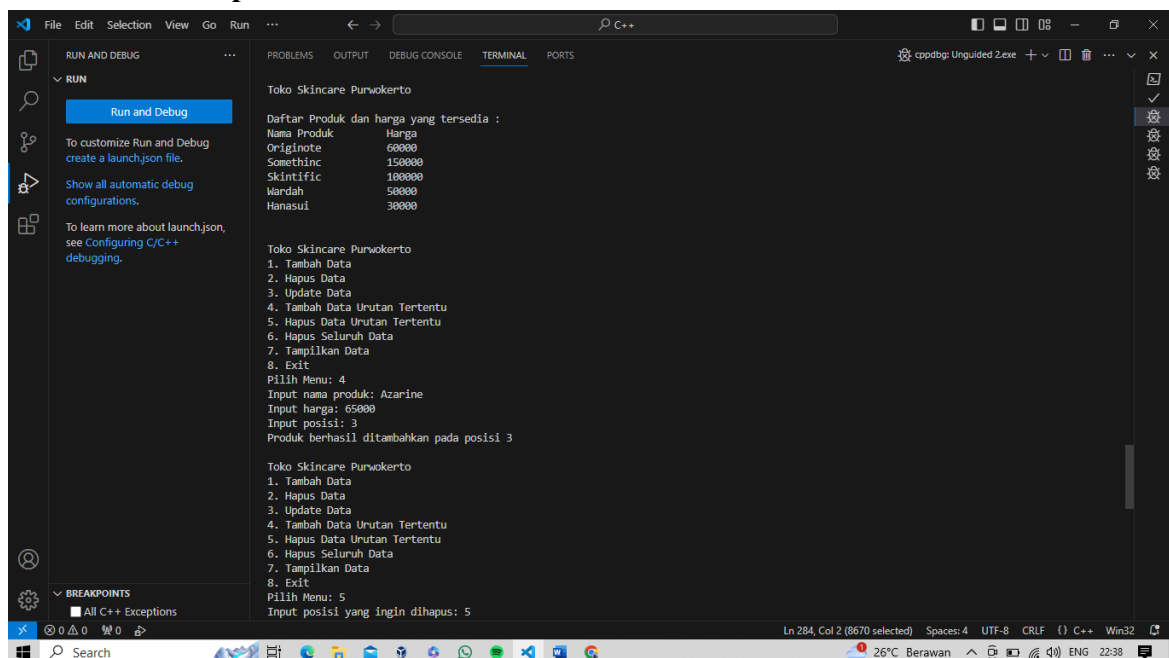
```

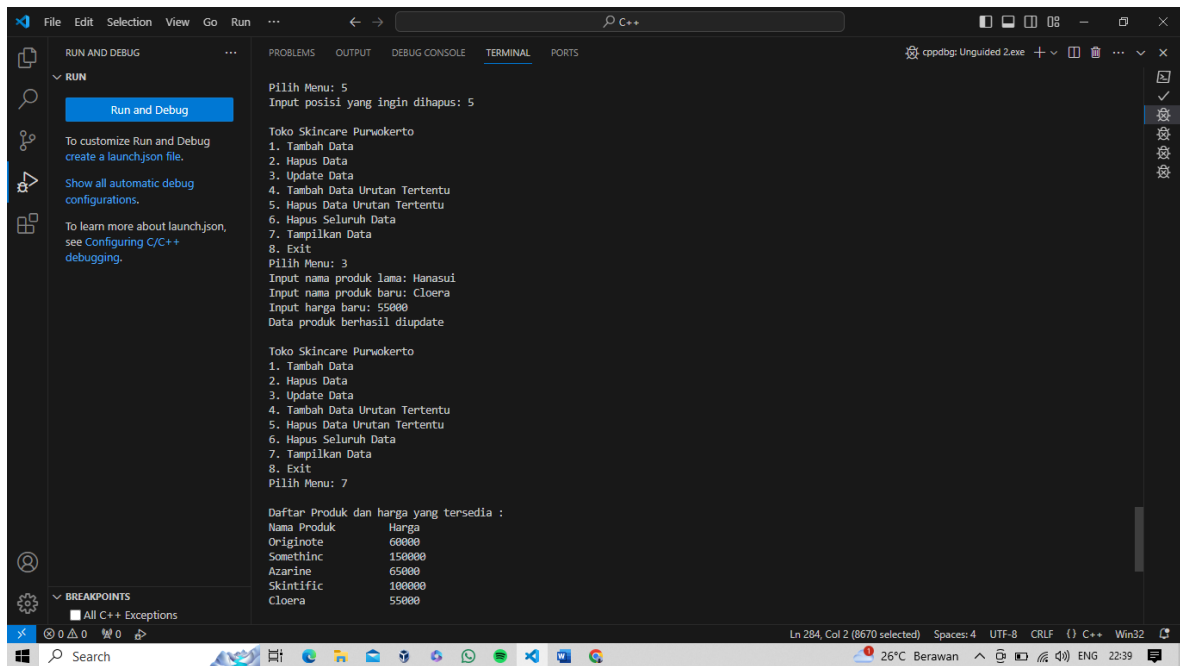
        cin >> position;
        list.hapusposisi(position);

        break;
    }
    // Memanggil fungsi hapus_semua
    case 6: {
        list.hapussemua();
        break;
    }
    // Memanggil fungsi display
    case 7: {
        list.tampilan();
        break;
    }
    case 8: {
        return 0;
    }
    default: {
        cout << "Input Invalid" << endl;
        break;
    }
    }
}
return 0;
}

```

Screenshots Output :





```
Pilih Menu: 5
Input posisi yang ingin dihapus: 5

Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih Menu: 3
Input nama produk lama: Manasui
Input nama produk baru: Cloera
Input harga baru: 55000
Data produk berhasil diupdate

Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih Menu: 7

Daftar Produk dan harga yang tersedia :
Nama Produk      Harga
Origino           60000
Somethinc         150000
Azarine           65000
Skintific         100000
Cloera            55000
```

Deskripsi :

Program di atas adalah sebuah program dalam bahasa C++ yang mengimplementasikan Doubly Linked List untuk menyimpan informasi tentang produk dan harga. Program ini memiliki dua kelas utama, yaitu kelas Node yang merepresentasikan setiap node dalam linked list, dan kelas DoublyLinkedList yang mengatur operasi-operasi pada linked list.

Kelas Node memiliki atribut-atribut untuk menyimpan nama produk, harga, serta pointer ke node sebelumnya dan node berikutnya dalam linked list. Sementara itu, kelas DoublyLinkedList memiliki pointer ke kepala (head) dan ekor (tail) dari linked list.

Program menyediakan fungsi-fungsi sebagai berikut:

1. tambahproduk(): Menambahkan produk baru ke dalam linked list di bagian atas.
2. hapusproduk(): Menghapus produk teratas dari linked list.
3. ubahproduk(): Mengubah data produk berdasarkan nama produk.
4. sisipposisi(): Menambahkan produk pada posisi tertentu dalam linked list.
5. hapusposisi(): Menghapus produk pada posisi tertentu dalam linked list.
6. hapussemua(): Menghapus seluruh data produk dari linked list.
7. tampilan(): Menampilkan seluruh data produk yang tersedia.

Program diawali dengan menambahkan beberapa produk awal ke dalam linked list dan menampilkan daftar produk. Selanjutnya, program memasuki loop tak terbatas yang memungkinkan pengguna memilih berbagai operasi yang ingin dilakukan pada linked list, seperti menambah, menghapus, mengubah, menampilkan, atau mengakhiri program. Setiap operasi yang dilakukan akan memperbarui dan menampilkan daftar produk sesuai dengan perubahan yang terjadi pada linked list.

D. Kesimpulan

Ketiga kode yang disajikan merupakan implementasi dari struktur data linked list dalam bahasa C++, masing-masing menawarkan fitur operasi dasar seperti penambahan, penghapusan, dan pengubahan data. Kode pertama menghadirkan single linked list non-circular dengan operasi seperti penambahan di depan, di belakang, dan di tengah, serta penghapusan pada posisi tertentu. Kode kedua dan ketiga menyajikan doubly linked list, dengan fitur tambahan seperti pengubahan data berdasarkan nama produk dan penambahan serta penghapusan pada posisi tertentu, dilengkapi dengan menu interaktif untuk pengguna. Keseluruhan, ketiga implementasi ini memberikan solusi yang berguna untuk pengelolaan data dengan struktur linked list yang sesuai dengan kebutuhan dan preferensi pengguna.

E. Referensi

Trivusi (2022) 'Struktur Data Linked List: Pengertian, Karakteristik, dan Jenis-jenisnya,' *Trivusi*, 16

September. <https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html>.

Teherag, S.D. (2020) 'Single Link List dan Double Link List,' *blogspot.com*, 3 March.

<https://stanleydaveteherag.blogspot.com/2020/03/single-link-list-dan-double-link-list.html>.

Fajri, D. (2017) 'LAPORAN PRAKTIKUM 3 ALGORITMA STRUKTUR DATA-SINGLE LINKED

LIST, DOUBLE LINKED LIST, CIRCULAR LINKED LIST,' *Universitasnegerimalang*

[Preprint].

https://www.academia.edu/32306667/LAPORAN_PRAKTIKUM_3_ALGORITMA_STRUKTUR_DATA_SINGLE_LINKED_LIST_DOUBLE_LINKED_LIST_CIRCULAR_LINKED_LIST.