LAPORAN PRAKTIKUM

MODUL 3
SINGLE AND DOUBLE LINKED LIST

Disusun oleh:
Ahmadan Syaridin
2311102038

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

1. Memahami perbedaan konsep Single dan Double Linked List
2. Mampu menerapkan Single dan Double Linked List ke Dalam pemrograman

BAB II

DASAR TEORI

1. **Single Linked List**

Single Linked List merupakan suatu bentuk struktur data yang berisi Kumpulan data yang disebut sebagai node yang tersusun secara sekuensial, saling sambung menyambung , dinamis, dan terbatas. Setiap elemen Dalam linked list dihubungkan ke elemen lain melalui pointer. Masing – masing komponen sering disebut juga dengan simpul atau node atau vertexs. Pointer adalah Alamat elemen dalam linked list dihubungkan ke elemen lain melalui pointer. Masing-masing komponen sering disebut dengan simpul atau node atau verteks. Pointer adalah alamat elemen. Setiap simpul pada dasarnya dibagi atas dua bagian pertama disebut bagian isi atau informasi atau data yang berisi nilai yang disimpan oleh simpul. Bagian kedua disebut bagian pointer yang berisi alamat dari node berikutnya atau sebelumnya. Dengan menggunakan struktur seperti ini, linked list dibentuk dengan cara menunjuk pointer next suatu elemen ke elemen yang mengikutinya. Pointer next pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut head dan elemen terakhir dari suatu list disebut tail.

Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List umumnya lebih efisien dalam penggunaan memori dibandingkan dengan jenis Linked List lainnya, seperti Double Linked List dan Circular Linked List. Single linked list yang kedua adalah circular linked list. Perbedaan circular linked list dan non circular linked adalah penunjuk next pada node terakhir pada circular linked list akan selalu merujuk ke node pertama.

2. **Double Linked List**

Double Linked List adalah struktur data Linked List yang mirip dengan Single Linked List, namun dengan tambahan satu pointer tambahan pada setiap simpul yaitu pointer prev yang menunjuk ke simpul sebelumnya. Dengan adanya pointer prev, Double Linked List memungkinkan untuk Melakukan operasi penghapusan dan penambahan pada simpul mana saja secara efisien. Setiap simpul pada Double Linked List memiliki tiga elemen penting, yaitu elemen data (biasanya berupa nilai), pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya.

Keuntungan dari Double Linked List adalah memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul dimana saja dengan efisien, sehingga sangat berguna dalam implementasi beberapa algoritma yang membutuhkan operasi tersebut. Selain itu, Double Linked List juga memungkinkan kita untuk melakukan traversal pada list baik dari depan (head) maupun dari belakang (tail) dengan mudah. Namun, kekurangan dari Double Linked List adalah penggunaan memori yang lebih besar dibandingkan dengan Single Linked List, karena setiap simpul membutuhkan satu pointer tambahan. Selain itu, Double Linked List juga membutuhkan waktu eksekusi yang lebih lama dalam operasi penambahan dan penghapusan jika dibandingkan dengan Single Linked List

BAB III

GUIDED

1. Guided

a) Latihan Single Linked List

Source code

```
#include <iostream>

using namespace std;

/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node
struct Node
{
    // komponen/member
    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;
    }
}
```

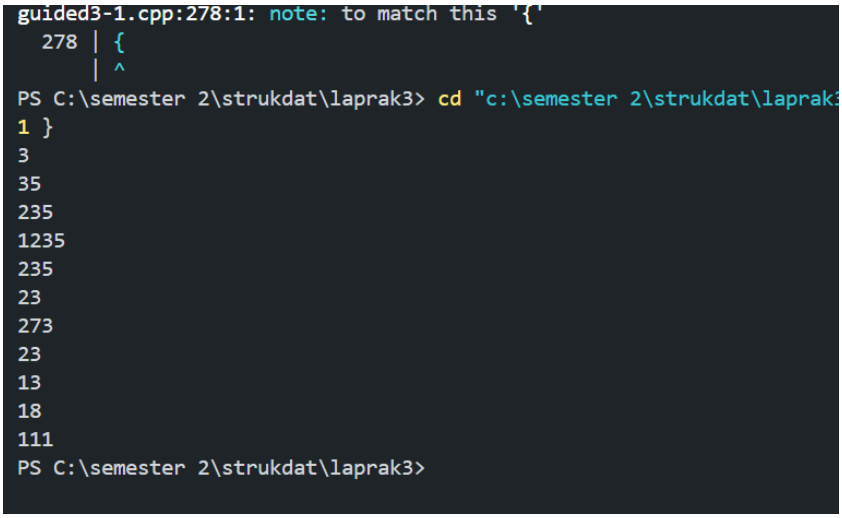
```
// Hapus Tengah
void hapusTengah(int posisi)
{
    Node *hapus, *bantu, *bantu2;
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi di luar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {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

Sreenshoot



Deskripsi program

- Struct Node : Mendeklarasikan struktur data Node.
- Data : Member/atribut untuk menyimpan data.
- Next : Member/atribut pointer yang menunjuk ke node berikutnya.
- Init (): Menginisialisasi list dengan membuat head dan tail menjadi NULL.
- Is Empty(): Mengecek apakah list kosong dengan memeriksa head.
- Inser tDepan(int nilai): Menambahkan node baru di depan list.
- Inser tBelakang(int nilai): Menambahkan node baru di belakang list.
- Hitung List(): Menghitung jumlah node dalam list.
- Inser tTengah(int data, int posisi): Menambahkan node baru di tengah list pada posisi tertentu.
- hapusDepan(): Menghapus node pertama dari list.
- hapusBelakang(): Menghapus node terakhir dari list.
- Hapus Tengah(int posisi): Menghapus node di tengah list pada posisi tertentu.
- Ubah Depan(int data): Mengubah data node pertama.
- Ubah tengah(int data, int posisi): Mengubah data node di tengah list pada posisi tertentu.
- Ubah Belakang(int data): Mengubah data node terakhir.
- Clear List(): Menghapus semua node dari list.
- Tampi l(): Menampilkan data semua node dalam list.

2. Guided 2

b) Latihan Double Linked List

Source code

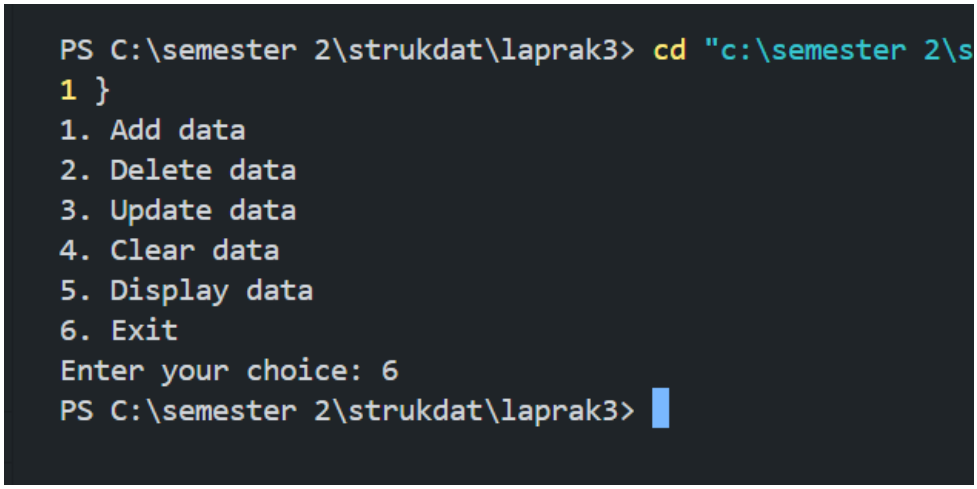
```
#include <iostream>

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;
}
```

Screenshot



Deskripsi program

Doubly Linked List adalah struktur data linear dimana setiap node memiliki 2 pointer, yaitu:

- next: pointer ke node selanjutnya
- prev: pointer ke node sebelumnya
-

Program ini menyediakan fungsi-fungsi untuk mengelola Doubly Linked List, yaitu:

- push(data): Menambahkan node baru berisi data di depan list.
- pop(): Menghapus node pertama dari list.
- update(oldData, newData): Mencari node dengan oldData dan memperbaharui datanya menjadi newData.
- deleteAll(): Menghapus semua node dari list.
- display(): Menampilkan data dari seluruh node dalam list.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

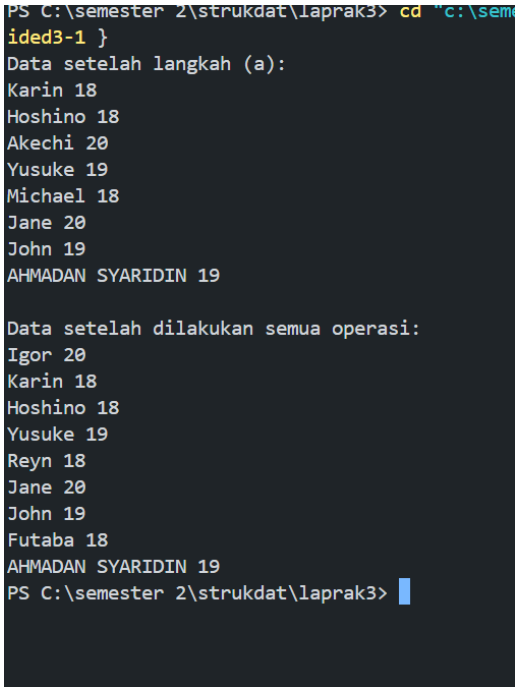
```cpp
#include <iostream>
using namespace std;

struct Node {
    string nama;
    int usia;
    Node* next;
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    void insertAwal(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = head;
        head = newNode;
    }

    void insertAkhir(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
```

```
            return;
        }

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

    void insertSetelah(string nama, int usia,
string namaSebelum) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;

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

        if (temp == nullptr) {
            cout << "Node dengan nama " <<
namaSebelum << " tidak ditemukan." << endl;
            return;
        }

        newNode->next = temp->next;
        temp->next = newNode;
    }

    void hapus(string nama) {
        if (head == nullptr) {
            cout << "Linked list kosong." << endl;
            return;
        }

        if (head->nama == nama) {
            Node* temp = head;
            head = head->next;
```

```
            delete temp;
            return;
        }

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

        if (temp == nullptr) {
            cout << "Node dengan nama " << nama <<
" tidak ditemukan." << endl;
            return;
        }

        prev->next = temp->next;
        delete temp;
    }

    void ubah(string nama, string namaBaru, int
usiaBaru) {
        Node* temp = head;
        while (temp != nullptr && temp->nama !=
nama) {
            temp = temp->next;
        }

        if (temp == nullptr) {
            cout << "Node dengan nama " << nama <<
" tidak ditemukan." << endl;
            return;
        }

        temp->nama = namaBaru;
        temp->usia = usiaBaru;
    }

    void tampilkan() {
```

```
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->nama << " " << temp->usia
<< endl;
            temp = temp->next;
        }
    }
};

int main() {
    LinkedList myList;

    myList.insertAwal("AHMADAN SYARIDIN", 19 );
    myList.insertAwal("John", 19);
    myList.insertAwal("Jane", 20);
    myList.insertAwal("Michael", 18);
    myList.insertAwal("Yusuke", 19);
    myList.insertAwal("Akechi", 20);
    myList.insertAwal("Hoshino", 18);
    myList.insertAwal("Karin", 18);

    cout << "Data setelah langkah (a):" << endl;
    myList.tampilkan();
    cout << endl;

    myList.hapus("Akechi");
    myList.insertSetelah("Futaba", 18, "John");
    myList.insertAwal("Igor", 20);
    myList.ubah("Michael", "Reyn", 18);
    cout << "Data setelah dilakukan semua operasi:"
<< endl;
    myList.tampilkan();

    return 0;
}
```

Screenshoot program



Deskripsi program

Program ini menyediakan class bernama LinkedList yang memiliki fungsi-fungsi untuk mengelola data:

- Insert Awal(string nama, int usia): Menambahkan node baru berisi nama dan usia di awal Linked List.
- Insert Akhir(string nama, int usia): Menambahkan node baru berisi nama dan usia di akhir Linked List.
- Insert Setelah(string nama, int usia, string namaSebelum): Menambahkan node baru berisi nama dan usia setelah node dengan namaSebelum.
- Hapus (string nama): Menghapus node dengan nama yang diberikan.
- Ubah (string nama, string namaBaru, int usiaBaru): Mencari node dengan nama dan mengubah datanya menjadi namaBaru dan usiaBaru.
- Tampilkan (): Menampilkan data dari seluruh node dalam Linked List, yaitu nama dan usia.

Pada fungsi main, program membuat objek myList dari class LinkedList. Kemudian program melakukan beberapa operasi pada myList sebagai contoh penggunaan:

- Memasukkan beberapa data (nama dan usia) ke dalam Linked List menggunakan insertAwal.
- Menampilkan isi Linked List setelah langkah awal (a).
- Menghapus node dengan nama "Akechi".
- Memasukkan node baru bernama "Futaba" setelah node bernama "John" menggunakan insertSetelah.
- Memasukkan node baru bernama "Igor" di awal Linked List menggunakan insertAwal.
- Mengubah data node dengan nama "Michael" menjadi "Reyn" menggunakan ubah.
- Menampilkan isi Linked List setelah semua operasi dilakukan.

2. Unguided 2

Source code

```
#include <iostream>
#include <string>

using namespace std;
struct Node
{
  string nama;
  int harga;
  Node *prev;
  Node *next;
};
class DoubleLinkedList
{
private:
  Node *head;
  Node *tail;
  int size;

public:
  DoubleLinkedList()
  {
    head = NULL;
    tail = NULL;
    size = 0;
  }
  void addData(string nama, int harga)
```

```
{
  Node *node = new Node;
  node->nama = nama;
  node->harga = harga;
  node->prev = tail;
  node->next = NULL;
  if (head == NULL)
  {
    head = node;
    tail = node;
  }
  else
  {
    tail->next = node;
    tail = node;
  }
  size++;
}
void addDataAt(int index, string nama, int harga)
{
  if (index < 0 || index > size)
  {
    cout << "Index tidak di temukan" << endl;
    return;
  }
  Node *node = new Node;
  node->nama = nama;
  node->harga = harga;
  if (index == 0)
  {
    node->prev = NULL;
    node->next = head;
    head->prev = node;
    head = node;
  }
  else if (index == size)
  {
    node->prev = tail;
    node->next = NULL;
    tail->next = node;
    tail = node;
```

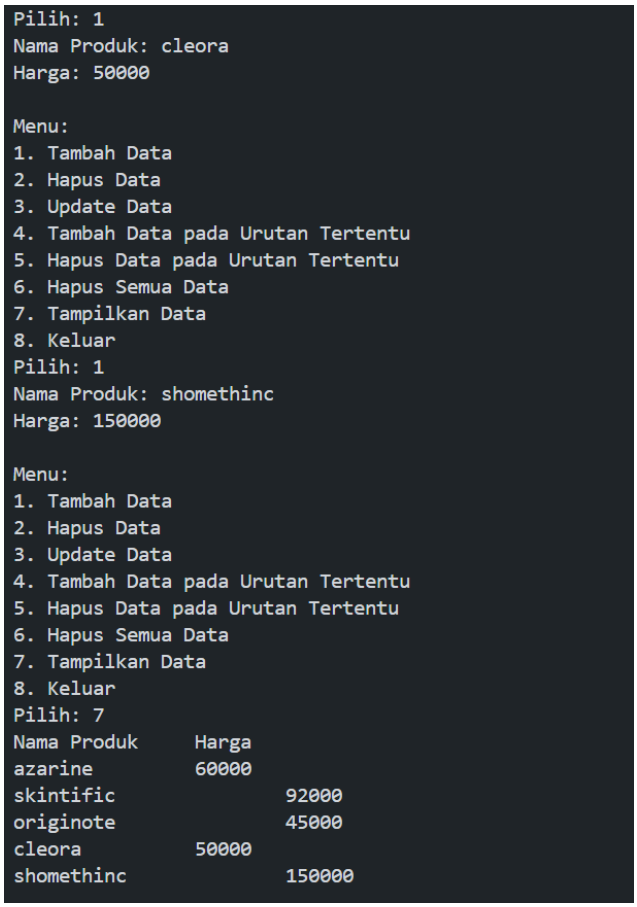
```
    }
    else
    {
      Node *current = head;
      for (int i = 0; i < index - 1; i++)
      {
        current = current->next;
      }
      node->prev = current;
      node->next = current->next;
      current->next->prev = node;
      current->next = node;
    }
    size++;
}
void deleteDataAt(int index)
{
    if (index < 0 || index >= size)
    {
      cout << "Index out of bounds" << endl;
      return;
    }

    if (index == 0)
    {
      Node *temp = head;
      head = head->next;
      head->prev = NULL;
      delete temp;
    }
    else if (index == size - 1)
    {
      Node *temp = tail;
      tail = tail->prev;
      tail->next = NULL;
      delete temp;
    }
    else
    {
      Node *current = head;
      for (int i = 0; i < index; i++)
```

```
      {
        current = current->next;
      }
      current->prev->next = current->next;
      current->next->prev = current->prev;
      delete current;
    }
    size--;
  }
  void clearData()
  {
    while (head != NULL)
    {
      Node *temp = head;
      head = head->next;
      delete temp;
    }
    tail = NULL;
    size = 0;
  }
  void displayData()
  {
    cout << "Nama Produk\tHarga" << endl;
    Node *current = head;
    while (current != NULL)
    {
      cout << current->nama << "\t\t" << current->harga <<
endl;
      current = current->next;
    }
  }
  void updateDataAt(int index, string nama, int harga)
  {
    if (index < 0 || index >= size)
    {
      cout << "Index out of bounds" << endl;
      return;
    }
    Node *current = head;
    for (int i = 0; i < index; i++)
    {
```

```
      current = current->next;
    }
    current->nama = nama;
    current->harga = harga;
  }
};
int main()
{
  DoubleLinkedList dll;
  int choice;
  string nama;
  int harga;
  int index;
  do
  {
    cout << "Menu:" << endl;
    cout << "1. Tambah Data" << endl;
    cout << "2. Hapus Data" << endl;
    cout << "3. Update Data" << endl;
    cout << "4. Tambah Data pada Urutan Tertentu" << endl;
    cout << "5. Hapus Data pada Urutan Tertentu" << endl;
    cout << "6. Hapus Semua Data" << endl;
    cout << "7. Tampilkan Data" << endl;
    cout << "8. Keluar" << endl;
    cout << "Pilih: ";
    cin >> choice;
    switch (choice)
    {
    case 1:
      cout << "Nama Produk: ";
      cin >> nama;
      cout << "Harga: ";
      cin >> harga;
      dll.addData(nama, harga);
      break;
    case 2:
      cout << "Index: ";
      cin >> index;
      dll.deleteDataAt(index);
      break;
    case 3:
```

```
      cout << "Index: ";
      cin >> index;
      cout << "Nama Produk: ";
      cin >> nama;
      cout << "Harga: ";
      cin >> harga;
      dll.updateDataAt(index, nama, harga);
      break;
    case 4:
      cout << "Index: ";
      cin >> index;
      cout << "Nama Produk: ";
      cin >> nama;
      cout << "Harga: ";
      cin >> harga;
      dll.addDataAt(index, nama, harga);
      break;
    case 5:
      cout << "Index: ";
      cin >> index;
      dll.deleteDataAt(index);
      break;
    case 6:
      dll.clearData();
      break;
    case 7:
      dll.displayData();
      break;
    case 8:
      break;
    default:
      cout << "Pilihan tidak valid" << endl;
      break;
    }
    cout << endl;
  } while (choice != 8);
  return 0;
}
```

Screenshot



```
Pilih: 1
Nama Produk: cleora
Harga: 50000

Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 1
Nama Produk: shomethinc
Harga: 150000

Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 7
Nama Produk     Harga
azarine         60000
skintific               92000
originote               45000
cleora          50000
shomethinc              150000
```

Deskripsi

Double Linked List adalah struktur data linear dimana setiap node (elemen) memiliki 2 buah pointer, yaitu:

- prev: pointer ke node sebelumnya
- next: pointer ke node selanjutnya

Program ini menyediakan class bernama DoubleLinkedList yang memiliki fungsi-fungsi untuk mengelola data produk:

- addData(string nama, int harga): Menambahkan data produk baru di akhir Linked List.
- addDataAt(int index, string nama, int harga): Menambahkan data produk baru pada posisi tertentu di Linked List.

- deleteDataAt(int index): Menghapus data produk pada posisi tertentu di Linked List.
- clearData(): Menghapus semua data produk dari Linked List.
- displayData(): Menampilkan seluruh data produk yang ada di Linked List.
- updateDataAt(int index, string nama, int harga): Memperbaharui data produk pada posisi tertentu di Linked List.

Selain fungsi-fungsi tersebut, program ini juga memiliki menu interaktif yang memungkinkan pengguna untuk

- Tambah Data (Menambahkan data produk baru di akhir Linked List)
- Hapus Data (Menghapus data produk pada posisi tertentu di Linked List)
- Update Data (Memperbaharui data produk pada posisi tertentu di Linked List)
- Tambah Data pada Urutan Tertentu (Menambahkan data produk baru pada posisi tertentu di Linked List)
- Hapus Data pada Urutan Tertentu (Menghapus data produk pada posisi tertentu di Linked List)
- Hapus Semua Data (Menghapus semua data produk dari Linked List)
- Tampilkan Data (Menampilkan seluruh data produk yang ada di Linked List)
- Keluar