PRAKTIKUM STRUKTUR DATA

MODUL 7 DAN 8

"SINGLE LINKED LIST dan DOUBLE LINKED LIST"

Disusun oleh :

Diana Eka Permata Sari

TRPL 2C

(362458302090)

PROGAM STUDI TEKNOLOGI REKAYASA PERANKAT LUNAK
JURUSAN BISNIS DAN INFORMATIKA

2025

Jalan Raya Jember No.KM13, Kawang, Labanasem, Kec. Kabat,
Kabupaten Banyuwangi, Jawa Timur 68461

Modul 7 Single Linked List

Tujuan :

1. Memahami konsep linked list
2. Memahami dan mampu membedakan linked list dengan array
3. Memahami konsep operasi yang ada pada linked list

Dasar Teori :

Linked list adalah sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian. Struktur berupa rangkaian elemen saling berkait dimana setiap elemen dihubungkan elemen lain melalui pointer. Pointer adalah alamat elemen.Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

Pembuatan Single Linked List dapat menggunakan 2 metode:

- LIFO (Last In First Out), aplikasinya : Stack (Tumpukan)
- FIFO (First In First Out), aplikasinya : Queue (Antrean)

Operasi-Operasi yang ada pada Linked List:

- Insert
  Istilah Insert berarti menambahkan sebuah simpul baru ke dalam suatu linked list.
- IsEmpty
  Fungsi ini menentukan apakah linked list kosong atau tidak.
- Find First
  Fungsi ini mencari elemen pertama dari linked list
- Find Next
  Fungsi ini mencari elemen sesudah elemen yang ditunjuk now
- Retrieve
  Fungsi ini mengambil elemen yang ditunjuk oleh now. Elemen tersebut lalu dikembalikan oleh fungsi.
- Update
  Fungsi ini mengubah elemen yang ditunjuk oleh now dengan isi dari sesuatu
- Delete Now
  Fungsi ini menghapus elemen yang ditunjuk oleh now. Jika yang dihapus adalahelemen pertama dari linked list (head), head akan berpindah ke elemen berikut.
- Delete Head
  Fungsi ini menghapus elemen yang ditunjuk head.Head berpindah ke elemen sesudahnya.
- Clear
  Fungsi ini menghapus linked list yang sudah ada. Fungsi ini wajib dilakukan bila anda ingin mengakhiri program yang menggunakan linked list. Jika anda melakukannya,

data- data yang dialokasikan ke memori pada program sebelumnya akan tetap tertinggal di dalam memori.

Tugas Pendahuluan :

1. Buatlah resume 1 halaman mengenai Single Linked List dan berikan penjelasannya
   Jawab :

   Single Linked List (SLL) adalah **struktur data linear dinamis** yang terdiri dari **rangkaian node**, di mana setiap node hanya memiliki satu arah, yaitu menunjuk ke **node berikutnya**. Berbeda dengan array, linked list memungkinkan penyisipan dan penghapusan elemen dengan efisien tanpa perlu memindahkan elemen lainnya.

   Operasi dasarnya :

   1. insertFirst() adalah tambah node di awal list
   2. insertLast() adalah tambah node di akhir list
   3. deletFirst() adalah hapus node paling depan
   4. printList() adalah menampilkan seluruh data dari awal ke akhir
   5. search(key) adalah mencari nilai dalam list

   Kelebihannya :

   1. Struktur fleksibel (dinamis).
   2. Penyisipan dan penghapusan awal cepat (O(1)).
   3. Tidak perlu alokasi ukuran tetap di awal

   Kekurangannya :

   1. Akses data lambat (O(n))
   2. Tidak mendukung akses indeks langsung.
   3. Implementasi lebih kompleks dibanding List bawaan Dart.

Percobaan :

1. Percobaan 1

```dart
class Node {
  int nodeValue;
  Node? next;

  Node(this.nodeValue) : next = null;
}

class SingleLinkedList {
  /* Single Linked List */
  Node? head = null;

```

```
  bool isEmpty() {
    return head == null;
  }

  void InsertFront(int data) {
    Node newNode = Node(data);
  if (isEmpty()) {
    head = newNode;
  } else {
    newNode.next = head;
    head = newNode;
  }
}

  void InsertBack(int data) {
    Node newNode = Node(data);
    if (isEmpty()) {
      head = newNode;
    } else {
      Node? temp = head;
      while (temp!.next != null) {
        temp = temp.next;
      }
      temp.next = newNode;
    }
  }

  void InsertAfter(Node prevNode, int newData) {
    if (prevNode == isEmpty()) {
      print("The given previous node cannot be null");
    }
    Node newNode = Node(newData);
    newNode.next = prevNode.next;
    prevNode.next = newNode;
  }

  void deleteFront() {
    if (!isEmpty()) {
      if (head!.next != null) {
        head = head!.next;
      } else {
        head = null;
      }
    }
  }

  void deleteEnd() {
    if (!isEmpty()) {
```

```
61.       if (head!.next != null) {
62.         Node? bantu = head;
63.         while (bantu!.next!.next != null) {
64.           bantu = bantu.next;
65.         }
66.         bantu.next = null;
67.       } else {
68.         head = null;
69.       }
70.     }
71.   }
72.
73.   void deleteMiddle(int cari) {
74.     if (!isEmpty()) {
75.       if (head!.next != null) {
76.         Node? bantu = head;
77.         while (bantu!.next!.nodeValue != cari) {
78.           bantu = bantu.next;
79.         }
80.         Node? hapus = bantu.next;
81.         bantu.next = hapus!.next;
82.         hapus.next = null;
83.       } else {
84.         head = null;
85.       }
86.     }
87.   }
88.
89.   void PrintList() {
90.     if (!isEmpty()) {
91.       Node? now = head;
92.       while (now != null) {
93.         print('${now.nodeValue} ');
94.         now = now.next;
95.       }
96.     } else {
97.       print('Linked List dalam kondisi kosong');
98.     }
99.   }
100.     }
101.     void main() {
102.        SingleLinkedList ll = SingleLinkedList();
103.
104.       print("Data:");
105.       ll.InsertFront(10);
106.       ll.InsertFront(30);
107.
108.       print("Operasi Insertion Front of Linked List:");
```

```
109.        ll.PrintList();
110.
111.        ll.InsertBack(5);
112.        print("Operasi Insertion Back of Linked List:");
113.        // ll.PrintList();
114.
115.        print("Insert After Data: ");
116.        ll.InsertAfter(ll.head!.next!.next!, 20);
117.        ll.PrintList();
118.
119.        print("Delete data:");
120.        // ll.deleteFront();
121.        ll.deleteEnd();
122.        ll.deleteMiddle(5);
123.         ll.PrintList();
124.      }
125.
```

Hasil run :

```
Data:
Operasi Insertion Front of Linked List:
30
10
Operasi Insertion Back of Linked List:
Insert After Data:
30
10
5
20
Delete data:
30
10

Exited.
```

Latihan :

1. Latihan no 3
   Buatlah fungsi untuk melakukan pembacaan Linked List! Node? findNode(int data) { }

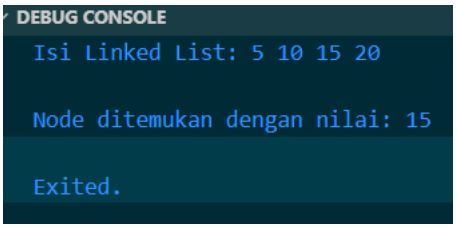
```
1. import 'dart:io';
2.
3. // NODE class untuk Single Linked List
4. class Node {
5.   int data;
6.   Node? next;
7.
8.   Node(this.data);
9. }
10.
11.// Single Linked List class
```

```
class SingleLinkedList {
  Node? head;

  // Menambahkan node di akhir list
  void insertEnd(int value) {
    Node newNode = Node(value);
    if (head == null) {
      head = newNode;
    } else {
      Node temp = head!;
      while (temp.next != null) {
        temp = temp.next!;
      }
      temp.next = newNode;
    }
  }

  // Fungsi membaca semua node (pembacaan maju)
  void printList() {
    Node? temp = head;
    stdout.write("Isi Linked List: ");
    while (temp != null) {
      stdout.write("${temp.data} ");
      temp = temp.next;
    }
    print("\n");
  }

  // Fungsi untuk mencari node dengan nilai tertentu
  Node? findNode(int data) {
    Node? current = head;
    while (current != null) {
      if (current.data == data) {
        return current;
      }
      current = current.next;
    }
    return null;
  }
}

void main() {
  SingleLinkedList list = SingleLinkedList();

  // Tambahkan data ke list
  list.insertEnd(5);
  list.insertEnd(10);
  list.insertEnd(15);
```

```
60.   list.insertEnd(20);
61.
62.   // Cetak list
63.   list.printList();
64.
65.   // Cari node dengan nilai tertentu
66.   int target = 15;
67.   Node? result = list.findNode(target);
68.
69.   if (result != null) {
70.     print("Node ditemukan dengan nilai: ${result.data}");
71.   } else {
72.     print("Node dengan nilai $target tidak ditemukan.");
73.   }
74. }
75.
76.
```

Fungsi printList() pada single linked list digunakan untuk membaca dan menampilkan isi list dari awal (head) hingga akhir (null). Ini dilakukan dengan menelusuri setiap node satu per satu menggunakan pointer sementara. Fungsi ini sangat penting untuk melihat data yang tersimpan dalam list secara berurutan.

Hasil run :



Modul 8 DOUBLE LINKED LIST

Tujuan :

1. Memahami konsep Double Linked List
2. Mengetahui cara membuat sebuah Node pada Double Linked List
3. Mampu membuat Double Linked List sendiri
4. Memahami operasi-operasi yang terdapat pada Double Linked List

Dasar Teori :

Double Linked List terdiri dari tiga bagian yaitu untuk menyimpan nilai dan dua reference yang menunjuk ke node selanjutnya (next node) dan node sebelumnya (previous node). Untuk bergerak maju dan mundur pada double linked list menggunakan link next dan prev pada node.

Pembacaan pada Double Linked List

Double Linked List dapat dibaca melalui dua arah.

- Pembacaan maju (forward scan) yaitu membaca double linked list dimulai dari reference front dan berakhir pada reference back.
- Pembacaan mundur (backward scan) yaitu membaca double linked list dimulai dari reference back dan berakhir pada reference front.

Representasi Node

Node pada Double Linked List direpresentasikan dengan class DNode. Kumpulan object DNode membentuk sebuah list disebut double linked list. Object DNode mempunyai tiga variabel:

- nodeValue untuk menyimpan nilai
- prev untuk menandai node sebelumnya
- Next untuk menandai node sesudahnya.

Class DNode mempunyai dua constructor.

- Default constructor
  Membuat object DNode dengan nodeValue bernilai null, sedangkan prev dan next diset dengan nilai this (link yang menunjuk ke dirinya sendiri) .
- Constructor dengan argument
  Untuk memberikan nilai pada nodeValue, sedangkan untuk variabel prev dan next diset dengan nilai this(link yang menunjuk ke dirinya sendiri)

Tugas Pendahuluan :

1. Buatlah review mengenai :
   - Double Linked List, Representasi Node pada Double Linked List, Cara menambahkan node di depan Double Linked List, dan cara menghapus node di depan Double Linked List

Jawab :

Double Linked List (DLL) adalah struktur data linier di mana setiap elemen (node) memiliki dua referensi/pointer:

- Satu menunjuk ke **node sebelumnya** (prev)
- Satu lagi menunjuk ke **node berikutnya** (next)

Keuntungan utama DLL dibanding Single Linked List adalah kemampuan untuk **menelusuri data dua arah**, baik maju maupun mundur.

Representasi node pada double linked list

```
class Node {
  int data;
  Node? prev;
  Node? next;

Node(this.data);
}
```

- data menyimpan nilai node
- prev menunjuk ke node sebelumnya
- next menunjuk ke node berikutnya

Menjadikan node baru sebagai head yang baru

itu ber arti :

1. Membuat node baru
2. Menautkan node baru ke head yang lama
3. Menjadikan node baru sebagai head yang baru

Contohnya yaitu :

```
class DoubleLinkedList {
  Node? head;

  void insertFront(int data) {
    Node newNode = Node(data);
    newNode.next = head;

    if (head != null) {
      head!.prev = newNode;
    }

    head = newNode;
  }
}
```

Menghapus node di depan double linked list

Itu ber arti :

1. Mengecek apakah list kosong
2. Mengatur head ke node berikutnya
3. Menghapus referensi dari node lama

Contohnya yaitu :

```
class DoubleLinkedList {
  Node? head;

  void deleteFront() {
    if (head == null) {
      print("List kosong");
      return;
    }

    head = head!.next;
    if (head != null) {
      head!.prev = null;
    }
  }
```

Percobaan :

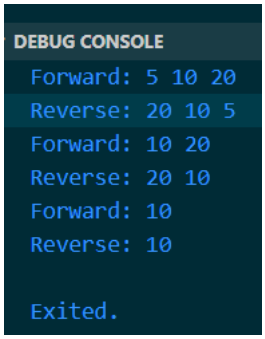
Percobaan 1 – 6

```
import 'dart:io';

class DNode<T> {
  T nodeValue;
  DNode<T>? prev;
  DNode<T>? next;

  DNode(this.nodeValue);
}

class DoublyLinkedList<T> {
  DNode<T>? head;

  bool isEmpty() => head == null;

  void append(T data) {
    DNode<T> newNode = DNode(data);
    if (isEmpty()) {
      head = newNode;
    } else {
      DNode<T> temp = head!;
```

```
    while (temp.next != null) {
      temp = temp.next!;
    }
    temp.next = newNode;
    newNode.prev = temp;
  }
}

void insertBeforeHead(T data) {
  DNode<T> newNode = DNode(data);
  if (isEmpty()) {
    head = newNode;
  } else {
    newNode.next = head;
    head!.prev = newNode;
    head = newNode;
  }
}

void deleteFront() {
  if (!isEmpty()) {
    if (head!.next != null) {
      DNode<T>? delete = head;
      head = head!.next;
      head!.prev = null;
      delete!.next = null;
    } else {
      head = null;
    }
  }
}

void deleteEnd() {
  if (!isEmpty()) {
    if (head!.next != null) {
      DNode<T>? temp = head;
      while (temp!.next != null) {
        temp = temp.next!;
      }
      temp.prev!.next = null;
      temp.prev = null;
    } else {
      head = null;
    }
  }
}

void printList() {
```

```
    DNode<T>? node = head;
    DNode<T>? tail;

    stdout.write("Forward: ");
    while (node != null) {
      stdout.write("${node.nodeValue} ");
      tail = node;
      node = node.next;
    }

    stdout.write("\nReverse: ");
    while (tail != null) {
      stdout.write("${tail.nodeValue} ");
      tail = tail.prev;
    }
    print('');
  }
}

void main() {
  DoublyLinkedList<int> list = DoublyLinkedList<int>();

  list.append(10);
  list.append(20);
  list.insertBeforeHead(5);
  list.printList();

  list.deleteFront();
  list.printList();

  list.deleteEnd();
  list.printList();
}
```

Hasil runnya :



Latihan :

3. Buatlah method untuk menambahkan Node di Double Linked List, setelah Node tertentu (pembacaan List menggunakan pembacaan maju)

void tambahNode_Setelah(DNode newNode, DNode target)

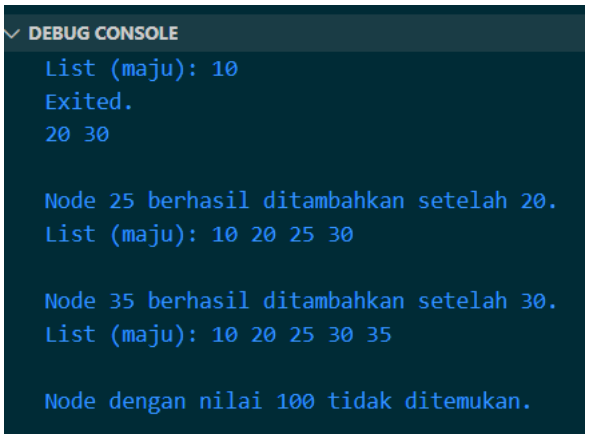
```
import 'dart:io';

class DNode<T> {
  T? nodeValue;
  DNode<T>? prev;
  DNode<T>? next;

  DNode.withValue(this.nodeValue);
}

class DoubleLinkedList<T> {
  DNode<T>? head;

  bool isEmpty() => head == null;

  // Menambahkan node di akhir
  void insertEnd(T item) {
    DNode<T> newNode = DNode.withValue(item);
    if (isEmpty()) {
      head = newNode;
    } else {
      DNode<T>? temp = head;
      while (temp!.next != null) {
        temp = temp.next;
      }
      temp.next = newNode;
      newNode.prev = temp;
    }
  }

  // Mencari node berdasarkan nilai
  DNode<T>? cariNode(T value) {
    DNode<T>? temp = head;
    while (temp != null) {
      if (temp.nodeValue == value) {
        return temp;
      }
      temp = temp.next;
    }
    return null; // tidak ditemukan
  }
```

```
  // Menambahkan node setelah node tertentu (target)
  void tambahNode_Setelah(T valueBaru, T valueTarget) {
    DNode<T>? target = cariNode(valueTarget);
    if (target == null) {
      print("Node dengan nilai $valueTarget tidak ditemukan.");
      return;
    }

    DNode<T> newNode = DNode.withValue(valueBaru);
    newNode.next = target.next;
    newNode.prev = target;

    if (target.next != null) {
      target.next!.prev = newNode;
    }
    target.next = newNode;

    print("Node $valueBaru berhasil ditambahkan setelah $valueTarget.");
  }

  // Menampilkan isi list maju
  void printList() {
    DNode<T>? temp = head;
    stdout.write("List (maju): ");
    while (temp != null) {
      stdout.write("${temp.nodeValue} ");
      temp = temp.next;
    }
    print("\n");
  }
}

void main() {
  var dll = DoubleLinkedList<int>();

  // Tambahkan beberapa node awal
  dll.insertEnd(10);
  dll.insertEnd(20);
  dll.insertEnd(30);
  dll.printList();

  // Tambahkan node setelah node bernilai 20
  dll.tambahNode_Setelah(25, 20);
  dll.printList();

  // Tambahkan node setelah node bernilai 30
  dll.tambahNode_Setelah(35, 30);
  dll.printList();
```

```
  // Coba tambahkan setelah node yang tidak ada
  dll.tambahNode_Setelah(99, 100); // node 100 tidak ada
}
```

Method tambahNode_Setelah() pada double linked list di Dart berfungsi untuk menyisipkan node baru **setelah node tertentu** dengan menjaga hubungan dua arah (prev dan next). Ini memungkinkan traversal dan manipulasi data secara efisien ke depan maupun ke belakang. Fungsi ini berguna dalam struktur data yang memerlukan fleksibilitas navigasi seperti undo-redo atau riwayat navigasi.

Hasil runnya :



Kesimpulan :

1. **Single Linked List (SLL)** hanya memiliki satu arah referensi (next), sehingga traversal hanya bisa dilakukan dari awal ke akhir. Struktur ini lebih sederhana dan efisien dalam penggunaan memori.
2. **Double Linked List (DLL)** memiliki dua arah referensi (next dan prev), memungkinkan traversal maju dan mundur. DLL lebih fleksibel untuk operasi seperti insert dan delete di kedua arah, namun membutuhkan lebih banyak memori karena ada dua pointer per node.

Referensi :

https://medium.com/@abdallahkshaf151/linked-list-in-dart-5a20614b7d0b

https://github.com/Diana0720/dianattugas.git