### KONSEP DASAR LINKED LIST

### **Definisi** : Linked list adalah struktur data berbasis node di mana setiap node berisi:

### **Data** : Dalam

### **Pointer** : Alamat yang menunjuk ke simpul berikutnya dalam memori.

* **Ciri Utama** :
* Elemen tidak disimpan secara berkala
* Penamaan

### JENIS-JENIS LINKED LIST

1. Daftar Tertaut Tunggal  
    Setia

* Contoh:head -> Node1 -> Node2-> NULL

1. Doubel linked list  
    Setiap node memiliki dua pointer: satu menunju

* Contoh: NULL<- Nodel 1<-<->Nodel 2<->Nodel 3->NULL

1. CIRCULAR LIKEND LIST:

Node terakhir terhubung kembali ke node pertama membentuk siklus.

Contoh: heed - > Node1 -> Node2 - > Node3 - > Head

### OPERASI PADA LINKED LIST

### Tambahan (Penyisipan ) : Elemen dapat ditambahkan:

* **Di awal** : Node baru menjadi kepala.
* **Di akhir** : Node baru ditambahkan setelah node terakhir.
* **Di posisi tertentu** : Node baru disisipkan di antara dua node.

1. Penghapusan (Penghapusan) :  
    Node d

* **Dari awal** : Kepala berpindah ke simpul berikutnya.
* **Dari akhir** : Node penunjuk kedu
* **Di posisi tertentu** : Node dihapus berd

1. Traversal :

proses mengunjungi atau mengakses setiap elemen dalam suatu struktur data secara berurutan.

### Keuntungan Linked List

* **Dinamis** : Ukuran dapat bertambah atau ber
* **Efisien dalam Penambahan/Penghapusan** : Tidak perlu menggeser

### Kekurangan Linked List

* **Akses Lambat** : Tidak memiliki indeks, sehingga memerlukan traversal untuk mengakses el
* **Memori Lebih** : Setiap node meme

### Kesimpulan dan Visualisasi

* Linked list adalah struktur data yang sangat berguna untuk aplikasi dinamis. Dengan ilustrasi dalam video ini, pembelajaran menjadi lebih mudah dipahami, khususnya untuk implementasi pointer.
* Video disertai contoh implementasi sederhana menggunakan kode C++.

1. PERBEDAAN LINKED LIST DENGAN TYPE DATA ARRAY
   * 1. Konsep Dasar

* **Susunan** :
* Struktur data yang menyimpan elemen secara berurutan dalam blok memori kontinu.
* Menggunakan indeks untuk mengakses langsung ke elemen.
* Cocok untuk data dengan jumlah tetap yang tidak sering dimodifikasi.
* **Daftar Tertaut** :
* Struktur data dinamis yang terdiri dari node-node individu, di mana setiap node memiliki dua komponen: data dan pointer ke node berikutnya.
* Elemen tidak disimpan secara berurutan di memori, melainkan dihubungkan dengan pointer.

### Perbandingan Operasi

* **Akses Data** :
* Array: Elemen akses menggunakan indeks langsung, waktu akses **O(1)** .
* Linked List: Memerlukan traversal dari awal hingga elemen yang diinginkan, waktu akses **O(n)** .
* **Penambahan dan Penghapusan Elemen** :
* Susunan:
* Penambahan atau penghapusan elemen memerlukan pergeseran elemen lainnya (O(n)).
* Tidak efisien untuk operasi yang sering berubah.
* Daftar Tertaut:
* Menambah atau menghapus cukup dengan memperbarui pointer, tanpa memindahkan elemen lainnya (O(1) atau O(n)).
* Lebih fleksibel untuk data yang sering dimodifikasi.

### Penggunaan Memori

* **Susunan** :
* Memori dialokasikan secara statistik.
* Risiko pemborosan memori jika elemen tidak terpakai, atau terjadi kegagalan jika kapasitas penuh.
* **Daftar Tertaut** :

Memori dipilih secara dinamis, sesuai kebutuhan.

* Membutuhkan memori tambahan untuk menyimpan pointer di setiap node.

### Contoh Implementasi

Video ini memberikan ilustrasi visual mengenai bagaimana array dan linked list menyimpan data, serta bagaimana elemen ditambahkan, dihapus, dan diakses.

* + 1. Kapan Menggunakan?
* **Susunan** :
* Cocok untuk data statistik dengan jumlah tetap.
* Digunakan jika akses cepat lebih diutamakan.
* **Daftar Tertaut** :
* Cocok untuk data dinamis dengan penambahan atau penghapusan yang sering.
* Digunakan jika ukuran data tidak dapat diprediksi.

### Kesimpulan

Array dan linked list memiliki kelebihan dan kekurangan masing-masing. Pemilihan struktur data tergantung pada kebutuhan aplikasi, apakah prioritas pada kecepatan akses atau halus dalam modifikasi data.

1. CONTOH PENGGUNAAN LINKED LIST

class Node:

def \_init\_(self, nama, nim):

self.nama = nama # Menyimpan nama mahasiswa

self.nim = nim # Menyimpan NIM mahasiswa

self.next = None # Pointer ke node berikutnya (default None)

class LinkedList:

def \_init\_(self):

self.head = None # Inisialisasi linked list kosong

def tambah\_mahasiswa(self, nama, nim):

"""Menambahkan mahasiswa baru ke akhir list"""

new\_node = Node(nama, nim)

if self.head is None:

self.head = new\_node # Jika list kosong, node pertama jadi head

else:

temp = self.head

while temp.next:

temp = temp.next

temp.next = new\_node # Menambahkan node baru ke akhir list

def tampilkan\_mahasiswa(self):

"""Menampilkan semua mahasiswa dalam list"""

if self.head is None:

print("Tidak ada data mahasiswa.")

return

temp = self.head

while temp:

print(f"Nama: {temp.nama}, NIM: {temp.nim}")

temp = temp.next

# Penggunaan Linked List

linked\_list = LinkedList()

# Menambahkan data mahasiswa

linked\_list.tambah\_mahasiswa("Andi", "20210101")

linked\_list.tambah\_mahasiswa("Budi", "20210202")

linked\_list.tambah\_mahasiswa("Citra", "20210303")

# Menampilkan semua data mahasiswa

linked\_list.tampilkan\_mahasiswa()

1. IMPLEMENTASI LINKED LIST DALAM BAHASA PEMOGRAMAN PYTHON

