





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 1

Pengantar Struktur Data

Struktur Data ?

- Struktur data adalah cara yang sistematis untuk mengorganisasi/mengatur dan mengakses data.
- Tujuan dari struktur data yaitu agar cara merepresentasikan data dalam program dapat dilakukan secara efisien di memori dan penyimpanan program ke storage lebih mudah dilakukan.

Hal-hal ?

- Saya dapat disemester 3 (Informatika)
- Lanjutan dari matkul DDP (Algoritma 1) dan Algoritma 2
- Implementasi menggunakan Bahasa pemrograman C++
- Konsep dan Implementasi berada di satu playlist ini.

Pra-syarat ?

- Sudah finish dan selesai DDP (Algoritma 1), Sorting & Searching pada Algoritma 2
- Dasar Pemrograman C++
- Software yang dibutuhkan
- Kuota
- Enjoy, siap & semangat

Tujuan Struktur Data ?

- Dapat memahami tentang pentingnya dan konsep dari struktur data beserta penerapannya untuk menyelesaikan masalah
- Dapat memahami tentang bagaimana penggunaan struktur data dalam algoritma maupun implementasi dalam pemrograman dengan bahasa pemrograman C++
- Memahami materi yang diberikan diplaylist ini.

Materi ?

1. Pointer
2. Array
3. Struct
4. Recursive Function
5. Linked List
6. Stack
7. Queue
8. Searching
9. Hashing
10. Sorting
11. Tree
12. Graph





Video Selanjutnya

SKS Dasar C++



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 2

Pointer (Penunjuk)

Memori & Alamat ?

- Sebelum memahami tentang pointer, sebaiknya memahami dulu tentang alamat dalam memori komputer.
- Lokasi pada memori komputer memiliki alamat dan menyimpan sebuah nilai.
- Alamat atau address yang dimaksud bernilai numerik (umumnya dalam bentuk *hexadecimal*) yang sulit untuk digunakan secara langsung oleh programmer.
- Setiap variabel yang dibuat pada program akan diberi lokasi di memori komputer. Nilai dari variabel sebenarnya disimpan pada alamat yang diberikan.
- Untuk mengetahui dimana data disimpan, pada C++ digunakan operator & (referensi).

Analogi Alamat & Memori ?



Memori
Indonesia (Wilayah / Daerah)

Alamat / Address

Rumah No.A100, Jl.Apa, Komplek Apa, Kecamatan
Apa, Kabupaten Apa, Provinsi Apa, Kode Pos Berapa

Variabel
Rumah Budi

Nilai / Value
Budi



Pointer ?

- Variabel biasa digunakan untuk menyimpan nilai.
- Variabel pointer digunakan untuk menyimpan alamat dari variabel lainnya.
- Pointer adalah representasi simbolik dari alamat.
- Pointer adalah fitur yang powerful pada C++ dibandingkan pada Bahasa pemrograman lain seperti Java atau Python.
- Pointer digunakan untuk mengakses memori dan memanipulasi alamat.
- Pointer berguna mentransfer data yang berkapasitas besar melalui suatu fungsi. Pointer sangat erat kaitannya dengan array, sehingga variabel pointer dapat menggantikan fungsi dan variabel array.

Analogi Alamat & Memori ?



Memori
Indonesia (Wilayah / Daerah)

Alamat / Address

Rumah No.A100, Jl.Apa, Komplek Apa, Kecamatan
Apa, Kabupaten Apa, Provinsi Apa, Kode Pos Berapa

Variabel
Rumah Budi

Nilai / Value
Budi



Analogi Variabel Pointer ?



Tampilkan Alamat Variabel ?

```
int nilai = 5;  
cout << "Alamatnya adalah : " << &nilai;
```

```
Alamatnya adalah : 0x61fe1c  
Terminal will be reused by tasks, press any key to close it.
```

Deklarasi Variabel Pointer ?

```
int nilai = 5;
// Deklarasi Pointer
/*
typeData *namaVarPtr;
typeData *namaVarPtr = &namaVar;
*/
int *ptrNilai1;
ptrNilai1 = &nilai;
// atau
int *ptrNilai2 = &nilai;
cout << "Isi prtNilai1 adalah : " <<
ptrNilai1 << endl;
cout << "Isi prtNilai2 adalah : " <<
ptrNilai2 << endl;
```

Isi prtNilai1 adalah : 0x61fe0c
Isi prtNilai2 adalah : 0x61fe0c

Terminal will be reused by tasks, press any

Operator Dereference/Indirection ?

- Operator dereference/indirection (*) digunakan untuk mendapatkan nilai pada alamat tertentu.
- Operator inditrection (*) adalah operator unary yang membutuhkan hanya satu operan.
- Operator inditrection (*) adalah komplemen dari operator alamat (&)
- Jika (&) akan menghasilkan alamat dari variabel lain, (*) akan menghasilkan isi nilai dari variabel lain.

Tampilkan Dereference/Indirection ?

```
int nilai = 10;
int *ptr = &nilai;
cout<<"nilai : "<<nilai<<endl;
cout<<"&nilai : "<<&nilai<<endl;
cout<<"ptr : "<<ptr<<endl;
cout<<"*ptr : "<<*ptr<<endl;
```

```
nilai : 10
&nilai : 0x61fe14
ptr : 0x61fe14
*ptr : 10
```

Terminal will be reused by tasks, press any key to close it.

Manipulasi nilai ?

```
int nilai = 10;
int *ptr = &nilai;
cout << "nilai : " << nilai << endl;

// manipulasi nilai
*ptr = 14;
cout << "nilai : " << nilai << endl;
cout << "*ptr : " << *ptr << endl;
```

```
nilai : 10
nilai : 14
*ptr : 14
```

Terminal will be reused by tasks, press any key to close it.

Kesimpulan ?

- Lokasi pada memori komputer memiliki alamat dan menyimpan data/nilai.
- Alamat atau address yang dimaksud benilai numerik (umumnya dalam bentuk hexadecimal).
- Pointer adalah variable yang menyimpan alamat memori dari variable lain
- Pointer digunakan untuk mengakses memori dan memanipulasi alamat.
- Tidak hanya mengakses dan memanipulasi alamat. Dengan pointer kita juga bisa memanipulasi nilai variabel lain.



Video Selanjutnya

Struct / Structure



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 3

Struct (Structure)



Masalah pada Array ?

- Array mampu menyimpan data dalam jumlah tertentu namun dengan tipe data yang sama.
- Bagaimana jika dibutuhkan penyimpanan data secara terkelompok dengan tipe data yang berbeda?.
- Kita dapat menggunakan dan membuat tipe data baru yang kita sebut sebagai structure atau biasa disingkat dengan struct.

Structure atau Struct ?

- Structure adalah kumpulan variabel dengan tipe data yang dapat berbeda-beda dengan satu nama.
- Structure mirip dengan class, bedanya jika class terdiri dari variabel/properti dan method/behavior, sedangkan structure hanya terdiri dari kumpulan variabel/property/data dengan beragam tipe data.
- Konsep dari structure mirip dengan array, perbedaannya hanya pada tipe datanya. Jika array hanya satu tipe data yang sama, sedangkan struct berbeda-beda tipe data.

Structure, New Data Type ?

- Pada umumnya ketika membuat tempat penyimpanan data, kita akan menggunakan tipe data yang telah terdefinisi seperti int, float, char, bool, dan lainnya.
- Ternyata kita dapat membuat sendiri tipe data yang dibutuhkan.
- Tipe data baru yang didefinisikan sendiri bisa dalam bentuk structure, class, dan lainnya.
- Structure dapat berisi variabel, array, atau bahkan tipe data baru lainnya juga seperti structure.

Deklarasi Structure ?

```
// Deklarasi Structure
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    . . .
};*/
*/
```

Deklarasi Var dengan Tipe Data Struct ?



```
// Cara 1 (Global)
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    . . .
}varName1, varName2, . . .;
*/
```

```
// Cara 2 (Lokal)
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    . . .
};

int main()
{
    // structName varName1, varName2, . . . ;
}
```

Akses Member Structure ?

- Untuk mengakses member dari structure, dapat digunakan operator akses member berupa period(.)

```
// structName varName1, varName2, ... ;  
  
// varName1.memberName1 = value;  
// cin >> varName1.memberName2;
```

Inisialisasi Member Structure ?

```
// Cara 1
/*
varName1.memberName1 = value1;
varName1.memberName2 = value2;
...
*/
// Cara 2
/*
struct Name varName;
varName = {val1, val2, ...};
*/
// Cara 3
/*
structName varName = {val1, val2, ...};
*/
```

Array dalam Member Structure ?

```
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1[n];
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    ...
};*/
*/
```

```
// Cara 1
/*
varName1.memberName1[0] = value1;
varName1.memberName1[n] = value2;
...
*/
// Cara 2
/*
varName1.memberName1 = {val1, valN};
...
*/
*/
```

Array dari Structure ?



```
// Cara 1 (Global)
/*
structName varName[n];
*/
int main()
{
    // Cara 2 (Lokal)
    /*
    structName varName[n];
    */
}
```

```
// Inisialisasi
/*
varName[n].memberName1 = val1;
varName[n].memberName2 = val2;
...
*/
```

Nested Structure ?

- Bisa dilakukan dengan dua cara :
 - Membuat dua structure berbeda
 - Membuat structure dalam structure

```
// Didalam Structure
/*
struct structName1{
    // komponen / member
    dataTypeMember1 memberName1[n];
    dataTypeMember2 memberName2;
    struct structName2{
        // komponen / member
        dataTypeMember1 memberName1;
        dataTypeMember2 memberName2;
    } varName;
};*/
*/
```

```
// Terpisah
/*
struct structName1{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
};

struct structName2{
    // komponen / member
    dataTypeMember1 memberName1[n];
    dataTypeMember2 memberName2;
    structName1 varName;
    . . .
};*/
*/
```

Structure di Parameter Fungsi ?

```
/*
void funcName(structName varName){
    // mau diapain
}

dataType funcName(structName varName){
    // mau diapain
    return ... ;
}

*/
```

Kesimpulan ?

- Structure adalah kumpulan variabel dengan tipe data yang dapat berbeda-beda dengan satu nama.
- Structure dapat berisi variabel, array, atau bahkan tipe data baru lainnya juga seperti structure.
- Untuk mengakses member dari structure, dapat digunakan operator akses member berupa period(.)
- Array structure merupakan kumpulan structure dengan panjang tertentu.
- Structure dalam structure disebut juga Nested Structure.
- Structure dapat digunakan sebagai parameter atau return dari fungsi.



Video Selanjutnya

Linked List





Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 3

Struct (Structure)



Masalah pada Array ?

- Array mampu menyimpan data dalam jumlah tertentu namun dengan tipe data yang sama.
- Bagaimana jika dibutuhkan penyimpanan data secara terkelompok dengan tipe data yang berbeda?.
- Kita dapat menggunakan dan membuat tipe data baru yang kita sebut sebagai structure atau biasa disingkat dengan struct.

Structure atau Struct ?

- Structure adalah kumpulan variabel dengan tipe data yang dapat berbeda-beda dengan satu nama.
- Structure mirip dengan class, bedanya jika class terdiri dari variabel/properti dan method/behavior, sedangkan structure hanya terdiri dari kumpulan variabel/property/data dengan beragam tipe data.
- Konsep dari structure mirip dengan array, perbedaannya hanya pada tipe datanya. Jika array hanya satu tipe data yang sama, sedangkan struct berbeda-beda tipe data.

Structure, New Data Type ?

- Pada umumnya ketika membuat tempat penyimpanan data, kita akan menggunakan tipe data yang telah terdefinisi seperti int, float, char, bool, dan lainnya.
- Ternyata kita dapat membuat sendiri tipe data yang dibutuhkan.
- Tipe data baru yang didefinisikan sendiri bisa dalam bentuk structure, class, dan lainnya.
- Structure dapat berisi variabel, array, atau bahkan tipe data baru lainnya juga seperti structure.

Deklarasi Structure ?

```
// Deklarasi Structure
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    . . .
};*/
*/
```

Deklarasi Var dengan Tipe Data Struct ?

```
// Cara 1 (Global)
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    . . .
}varName1, varName2, . . .;
*/
```

```
// Cara 2 (Lokal)
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    . . .
};

int main()
{
    // structName varName1, varName2, . . . ;
}
```

Akses Member Structure ?

- Untuk mengakses member dari structure, dapat digunakan operator akses member berupa period(.)

```
// structName varName1, varName2, ... ;  
  
// varName1.memberName1 = value;  
// cin >> varName1.memberName2;
```

Inisialisasi Member Structure ?

```
// Cara 1
/*
varName1.memberName1 = value1;
varName1.memberName2 = value2;
...
*/
// Cara 2
/*
struct Name varName;
varName = {val1, val2, ...};
*/
// Cara 3
/*
structName varName = {val1, val2, ...};
*/
```

Array dalam Member Structure ?

```
/*
struct structName{
    // komponen / member
    dataTypeMember1 memberName1[n];
    dataTypeMember2 memberName2;
    dataTypeMember3 memberName3;
    ...
};*/
*/
```

```
// Cara 1
/*
varName1.memberName1[0] = value1;
varName1.memberName1[n] = value2;
...
*/
// Cara 2
/*
varName1.memberName1 = {val1, valN};
...
*/
*/
```

Array dari Structure ?



```
// Cara 1 (Global)
/*
structName varName[n];
*/
int main()
{
    // Cara 2 (Lokal)
    /*
    structName varName[n];
    */
}
```

```
// Inisialisasi
/*
varName[n].memberName1 = val1;
varName[n].memberName2 = val2;
...
*/
```

Nested Structure ?

- Bisa dilakukan dengan dua cara :
 - Membuat dua structure berbeda
 - Membuat structure dalam structure

```
// Didalam Structure
/*
struct structName1{
    // komponen / member
    dataTypeMember1 memberName1[n];
    dataTypeMember2 memberName2;
    struct structName2{
        // komponen / member
        dataTypeMember1 memberName1;
        dataTypeMember2 memberName2;
    } varName;
};*/
*/
```

```
// Terpisah
/*
struct structName1{
    // komponen / member
    dataTypeMember1 memberName1;
    dataTypeMember2 memberName2;
};

struct structName2{
    // komponen / member
    dataTypeMember1 memberName1[n];
    dataTypeMember2 memberName2;
    structName1 varName;
    . . .
};*/
*/
```

Structure di Parameter Fungsi ?

```
/*
void funcName(structName varName){
    // mau diapain
}

dataType funcName(structName varName){
    // mau diapain
    return ... ;
}

*/
```

Kesimpulan ?

- Structure adalah kumpulan variabel dengan tipe data yang dapat berbeda-beda dengan satu nama.
- Structure dapat berisi variabel, array, atau bahkan tipe data baru lainnya juga seperti structure.
- Untuk mengakses member dari structure, dapat digunakan operator akses member berupa period(.)
- Array structure merupakan kumpulan structure dengan panjang tertentu.
- Structure dalam structure disebut juga Nested Structure.
- Structure dapat digunakan sebagai parameter atau return dari fungsi.



Video Selanjutnya

Linked List





Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 4A

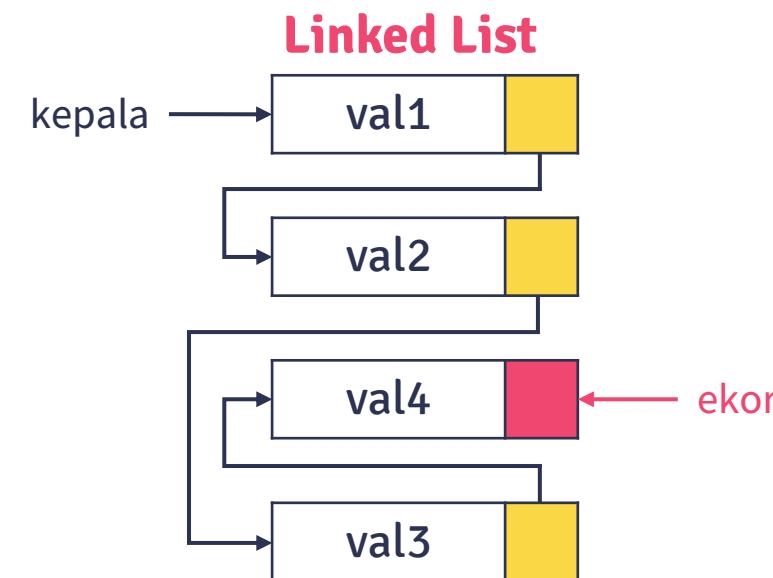
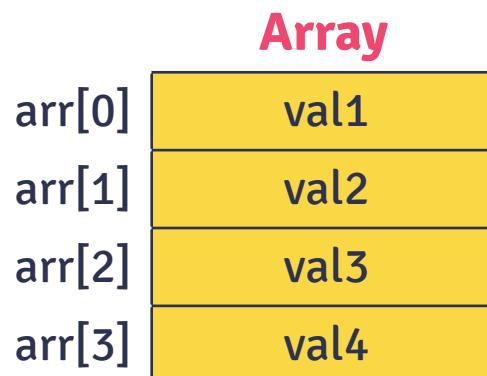
Single Linked List

Linked List ?

- Linked List adalah elemen yang berurutan yang dihubungkan dengan pointer.
- Elemen terakhir menunjuk ke NULL (untuk Linked List non Circular).
- Elemen pada Single Linked List dapat bertambah atau berkurang (dinamis) selama program dijalankan.
- Dapat dibuat selama diperlukan (hingga memori sistem habis).
- Linked List tidak membuang ruang memori (tetapi membutuhkan beberapa memori ekstra untuk pointer).

Array vs Linked List ?

- Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan Linked bisa meng-Allokasi memori secara dinamis.



Array vs Linked List ?

- Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan Linked bisa meng-Allokasi memori secara dinamis.

Array

v1	v2	v3	v4	v5	v6				
----	----	----	----	----	----	--	--	--	--

Tambahin valueN ke posisi ke-2

Array

v1	v2	v3	v4	v5		v6			
----	----	----	----	----	--	----	--	--	--

Array

v1	v2	v3	v4		v5	v6			
----	----	----	----	--	----	----	--	--	--

Array

v1	v2	v3		v4	v5	v6			
----	----	----	--	----	----	----	--	--	--

Array

v1	v2		v3	v4	v5	v6			
----	----	--	----	----	----	----	--	--	--

Array

v1		v2	v3	v4	v5	v6			
----	--	----	----	----	----	----	--	--	--

Array

v1	vN	v2	v3	v4	v5	v6			
----	----	----	----	----	----	----	--	--	--

Array vs Linked List ?

- Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan Linked bisa meng-Allokasi memori secara dinamis.

Linked List

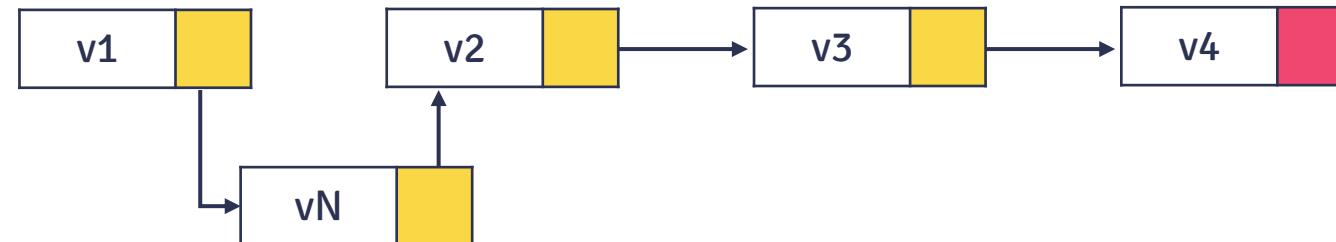


Tambahin valueN ke posisi ke-2

Linked List



Linked List

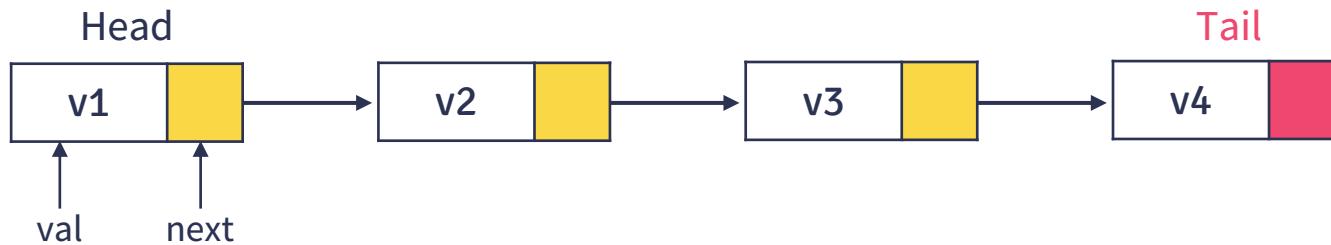


Tipe Linked List ?

- Single Linked List (Singly Linked List).
- Double Linked List (Doubly Linked List).
- Circular Linked List.
- Multiple Linked List.

Single Linked List ?

- Single Linked List merupakan suatu linked list yang hanya memiliki satu variabel pointer saja. Dimana pointer tersebut menunjuk ke node selanjutnya dan pointer pada tail menunjuk ke NULL.
- Navigasi item maju saja.
- Single Linked List terdiri dari sejumlah elemen (node) dimana setiap node memiliki penunjuk berikutnya ke elemen (node) berikutnya.
- Penunjuk node terakhir adalah NULL, yang menunjukkan akhir dari Single Linked List.



Deklarasi & Inisialisasi ?

```
/*
struct LinkListName{
    // komponen / member
    dataTypeData1 dataName1;
    ...
    LinkListName *next;
};
*/
```

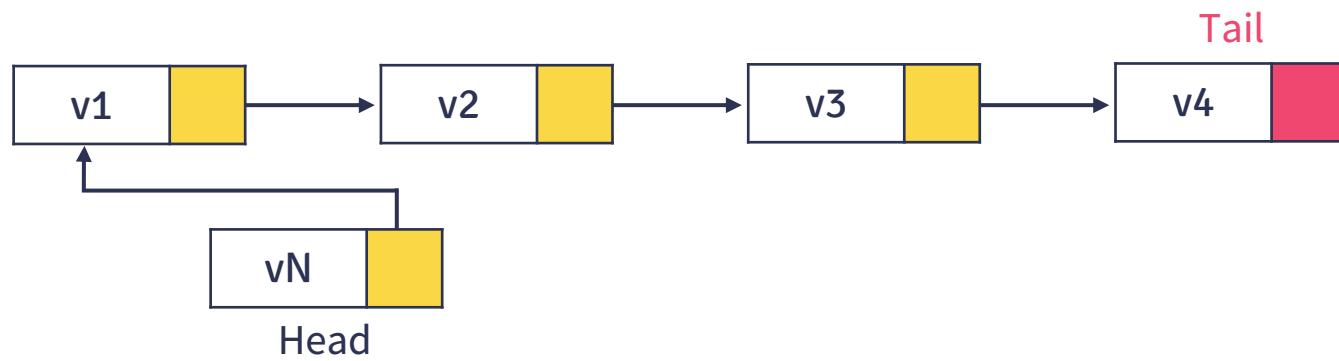
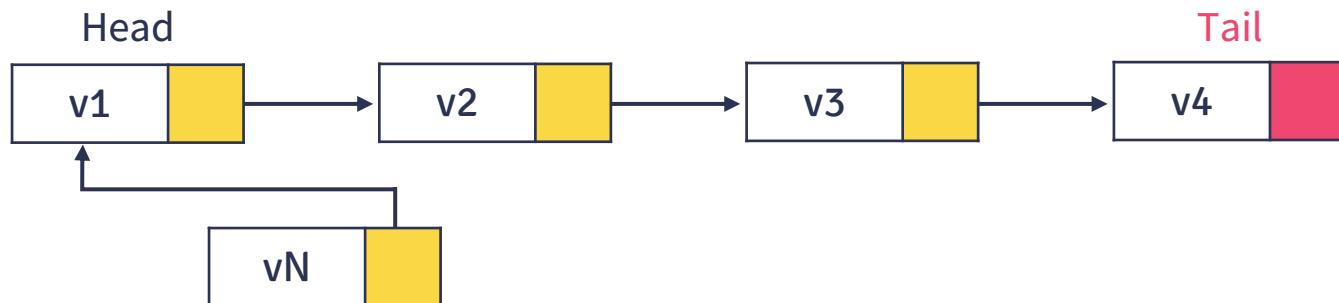
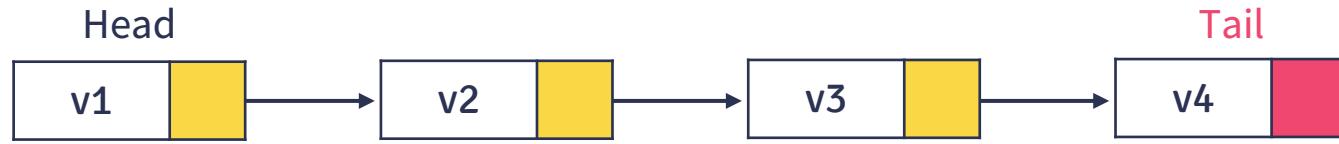
```
int main()
{
    /*
        LinkListName *node1, *nodeN;
        node1 = (LinkListName*) malloc(sizeof(LinkListName));
        nodeN = new LinkListName();
    */
    /*
        node1->dataName1 = valData1;
        ...
        node1->next = nodeN;

        nodeN->dataNameN = valDataN;
        ...
        nodeN->next = NULL;
    */
}
```

Print Single Linked List ?

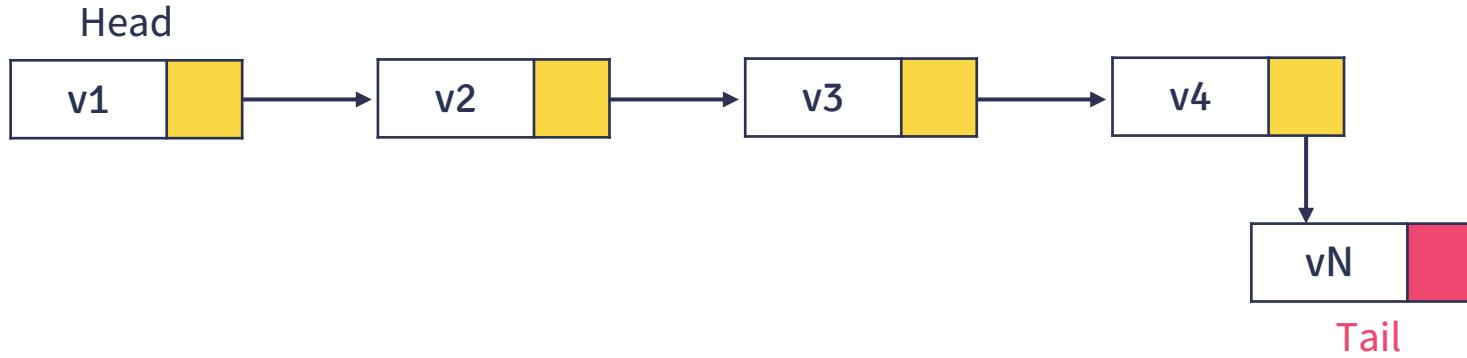
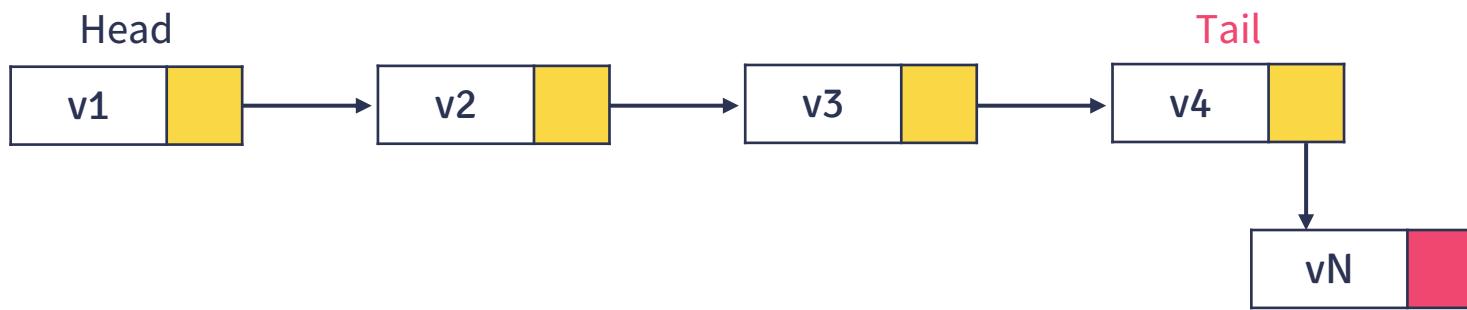
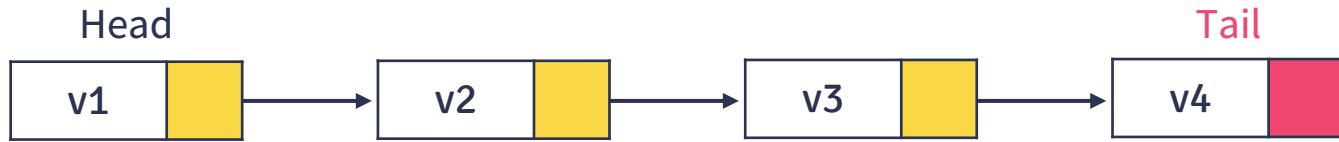
```
/*
LinkListName *cur;
cur = node1;
while( cur != NULL ){
    // print
    cur = cur->next;
}
*/
```

Added at Beginning Node ?

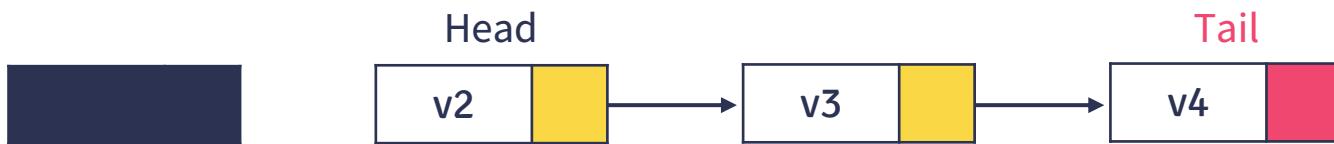
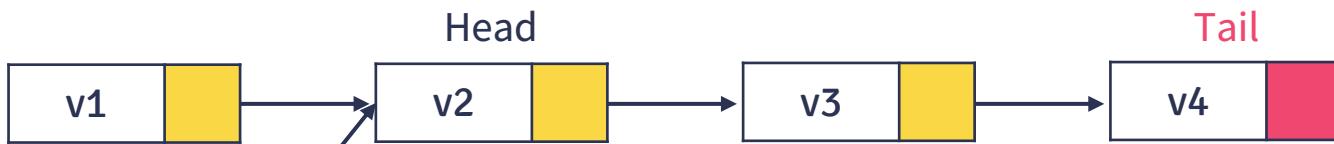
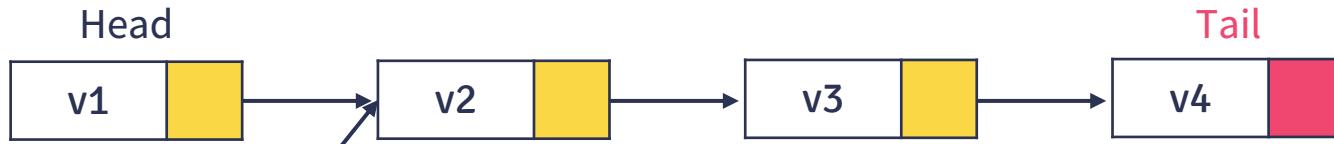


Head

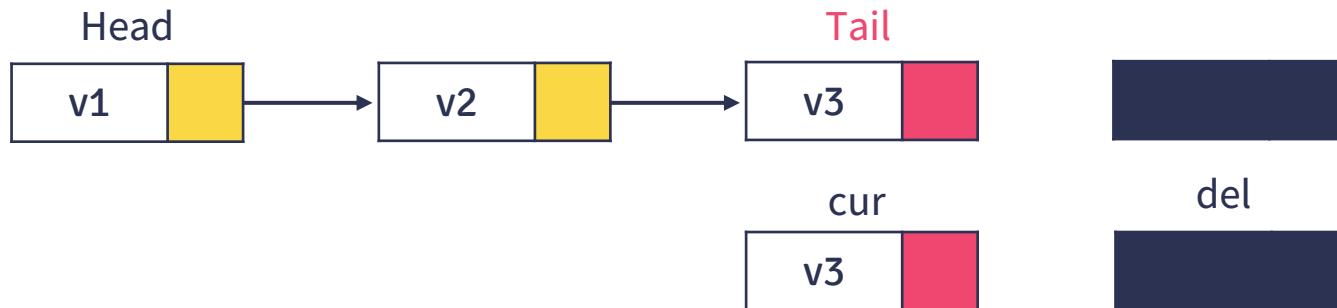
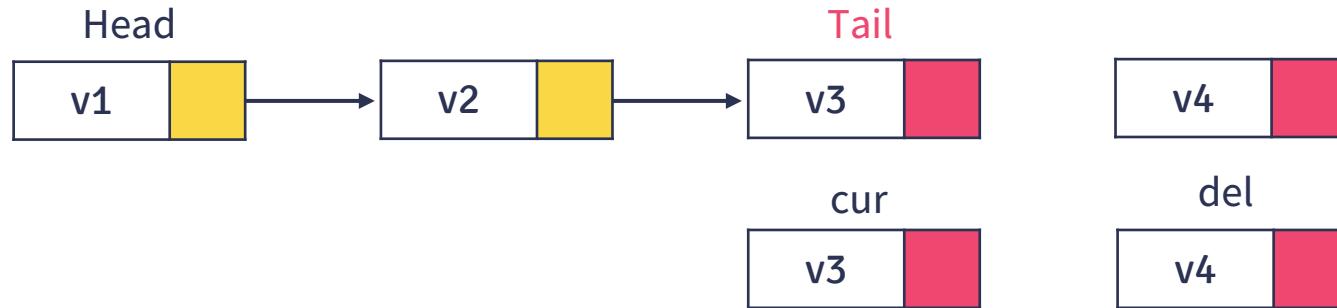
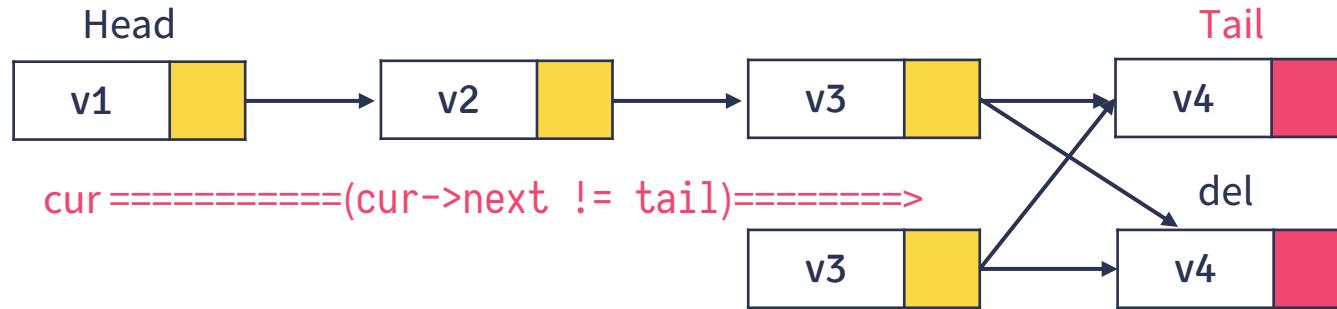
Added at Last Node ?



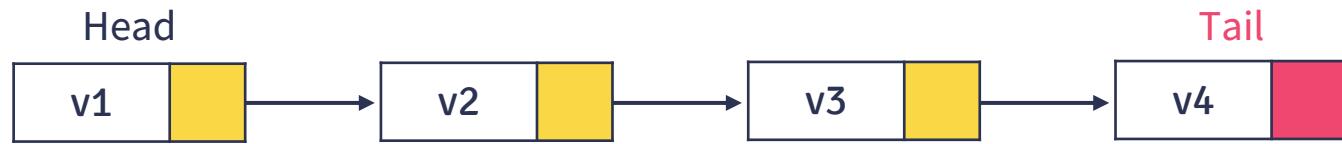
Delete the First Node ?



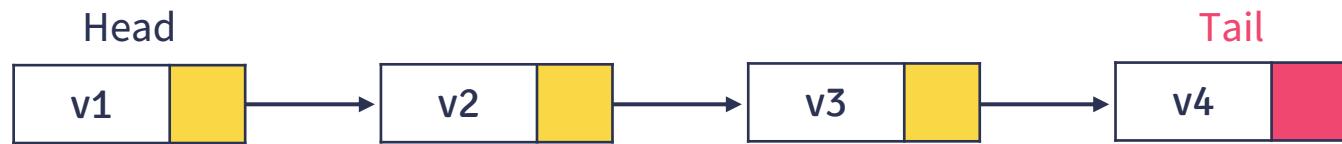
Delete the Last Node ?



Added at Middle Node ?



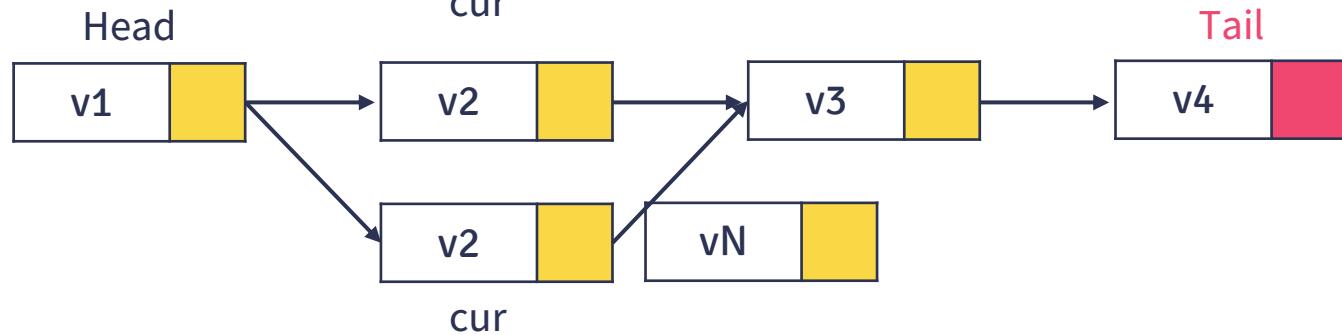
Tambah vN ke posisi 3



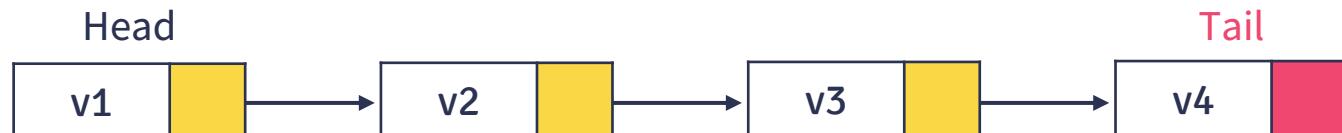
$\Rightarrow \text{loop } (1 < \text{posisi}-1) \Rightarrow$



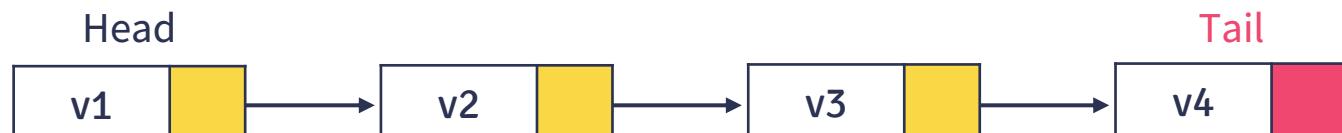
cur



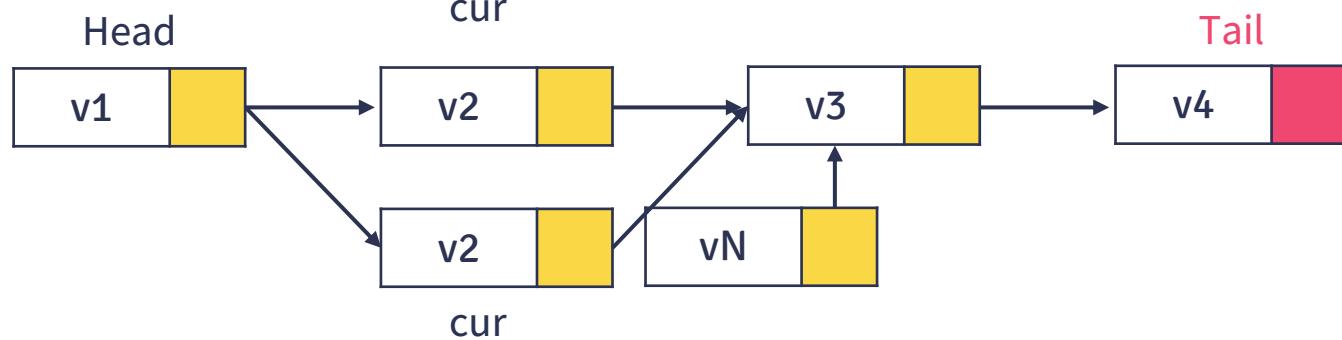
Added at Middle Node ?



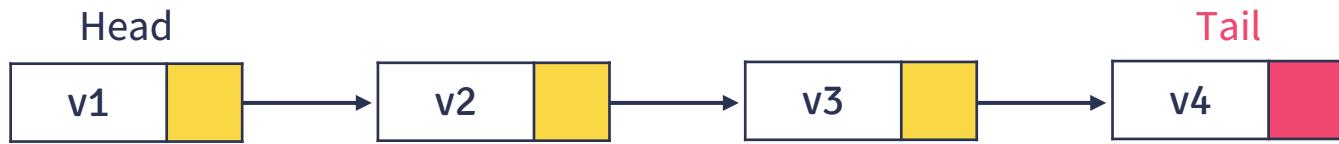
Tambah vN ke posisi 3



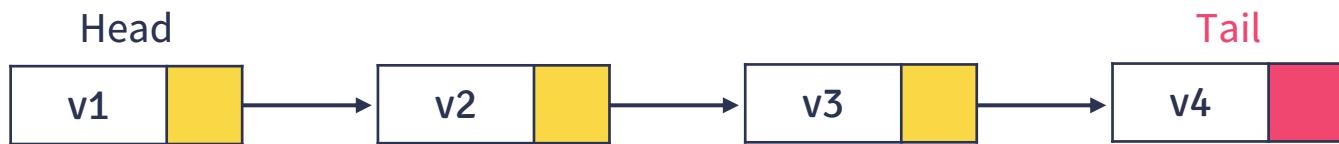
$\Rightarrow \text{loop } (1 < \text{posisi}-1) \Rightarrow$



Added at Middle Node ?



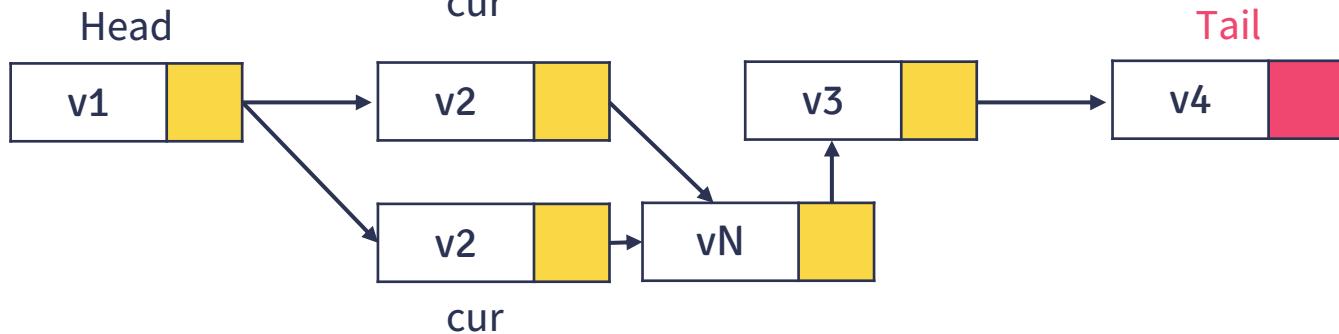
Tambah vN ke posisi 3



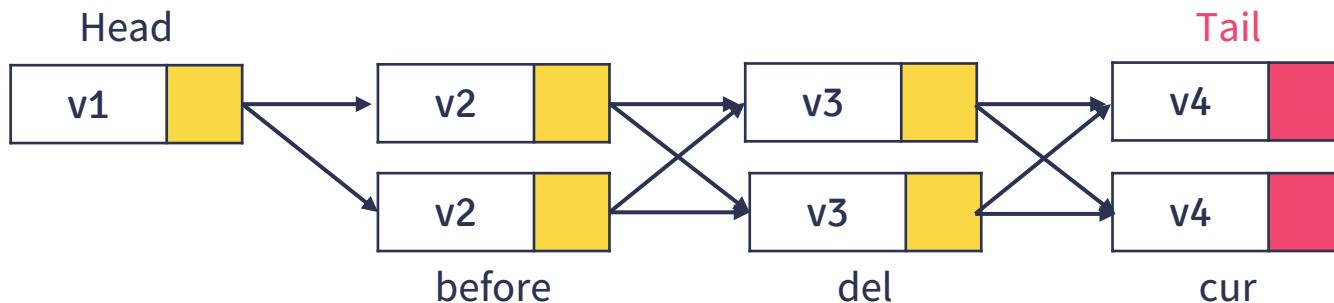
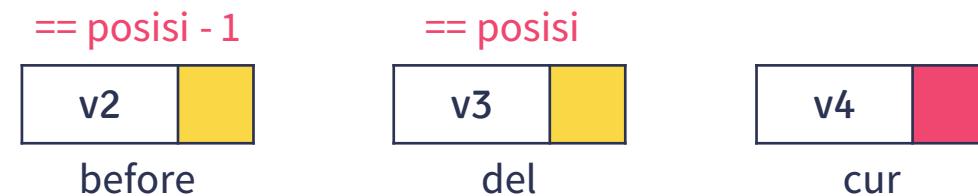
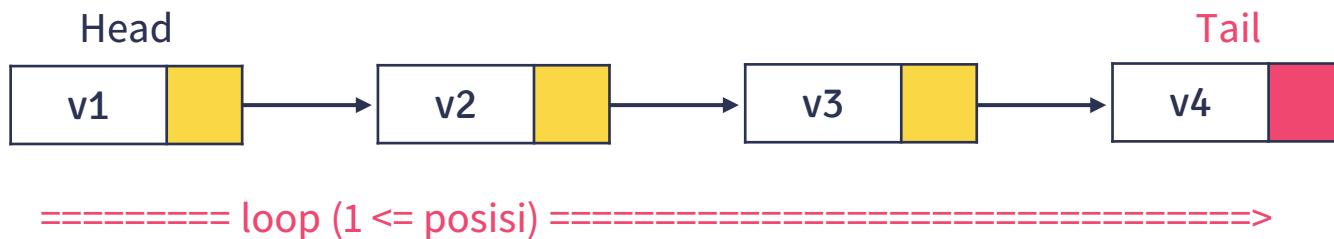
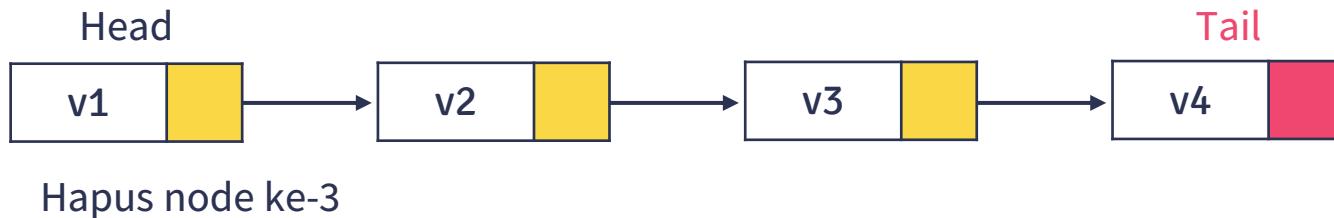
$\Rightarrow \text{loop } (1 < \text{posisi}-1) \Rightarrow$



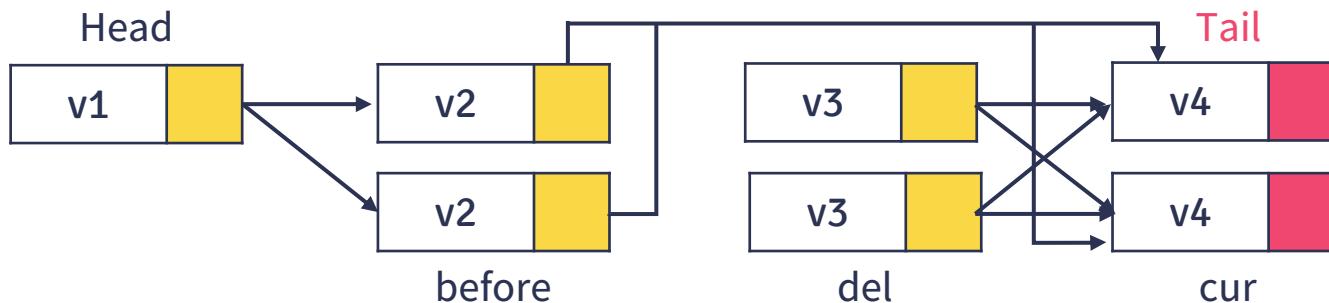
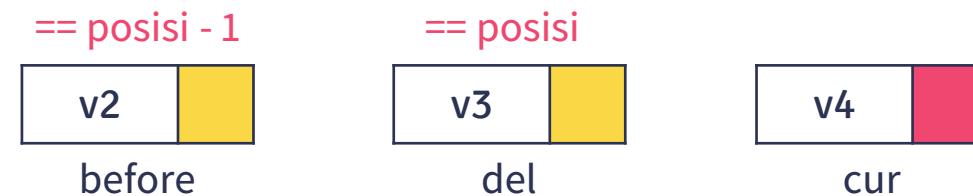
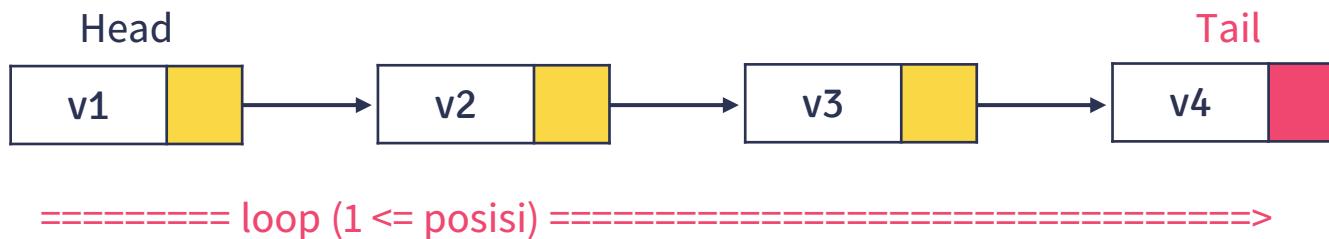
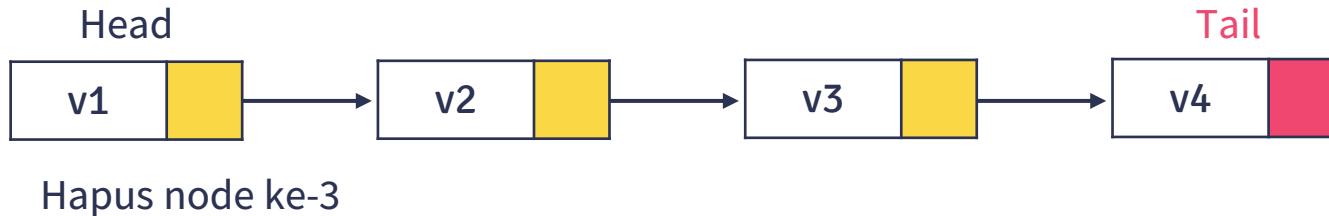
cur



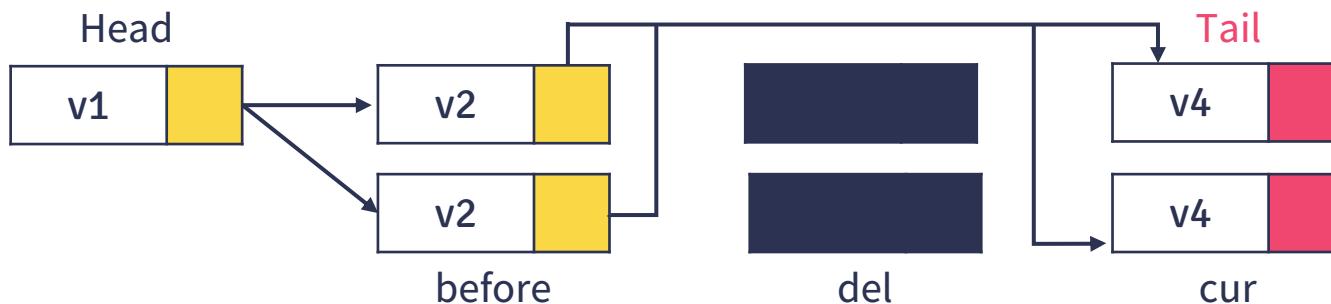
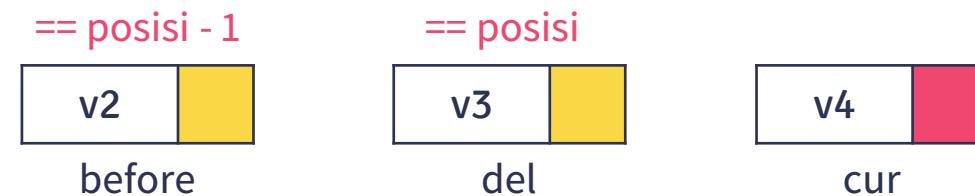
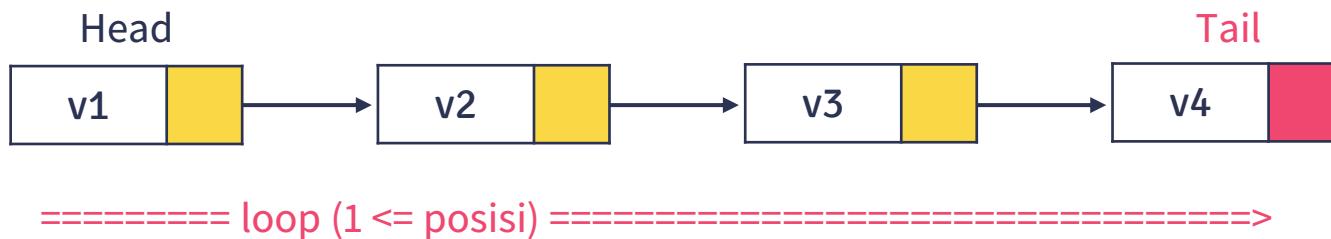
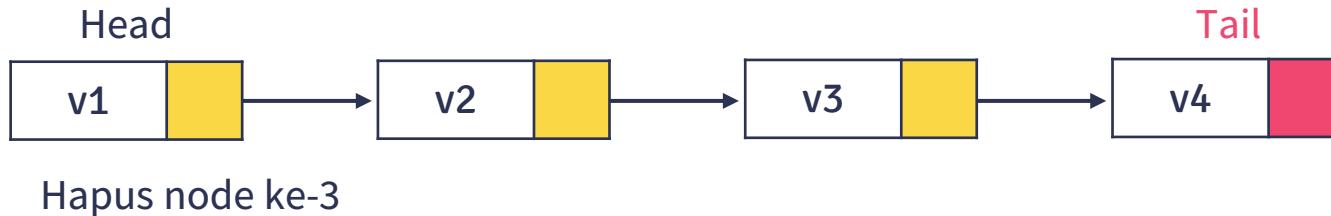
Delete at Middle Node ?



Delete at Middle Node ?



Delete at Middle Node ?





Video Selanjutnya

Double Linked List



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 4A

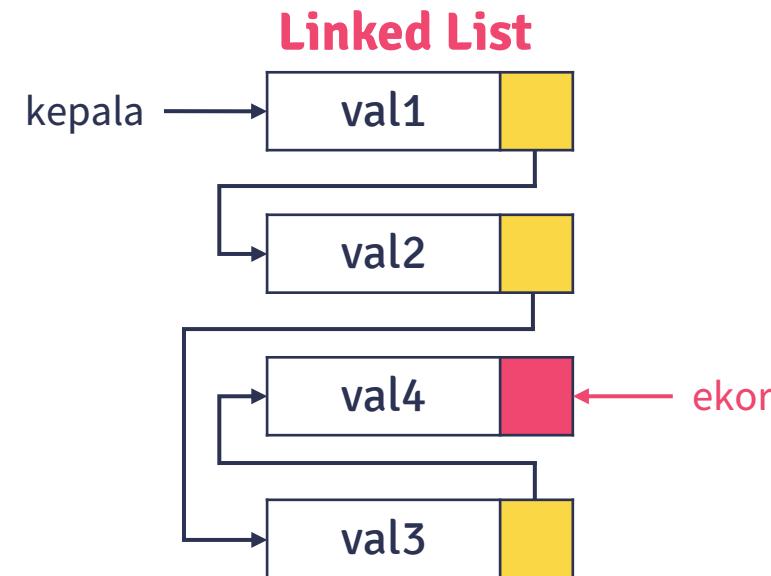
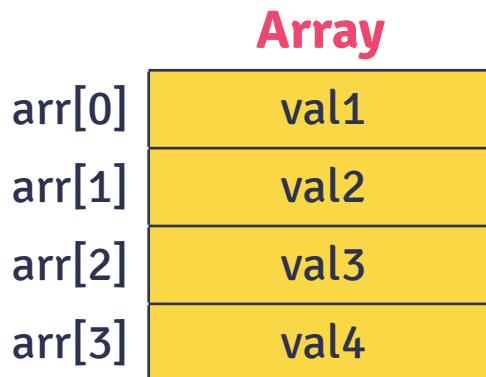
Single Linked List

Linked List ?

- Linked List adalah elemen yang berurutan yang dihubungkan dengan pointer.
- Elemen terakhir menunjuk ke NULL (untuk Linked List non Circular).
- Elemen pada Single Linked List dapat bertambah atau berkurang (dinamis) selama program dijalankan.
- Dapat dibuat selama diperlukan (hingga memori sistem habis).
- Linked List tidak membuang ruang memori (tetapi membutuhkan beberapa memori ekstra untuk pointer).

Array vs Linked List ?

- Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan Linked bisa meng-Allokasi memori secara dinamis.



Array vs Linked List ?

- Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan Linked bisa meng-Allokasi memori secara dinamis.

Array

v1	v2	v3	v4	v5	v6				
----	----	----	----	----	----	--	--	--	--

Tambahin valueN ke posisi ke-2

Array

v1	v2	v3	v4	v5		v6			
----	----	----	----	----	--	----	--	--	--

Array

v1	v2	v3	v4		v5	v6			
----	----	----	----	--	----	----	--	--	--

Array

v1	v2	v3		v4	v5	v6			
----	----	----	--	----	----	----	--	--	--

Array

v1	v2		v3	v4	v5	v6			
----	----	--	----	----	----	----	--	--	--

Array

v1		v2	v3	v4	v5	v6			
----	--	----	----	----	----	----	--	--	--

Array

v1	vN	v2	v3	v4	v5	v6			
----	----	----	----	----	----	----	--	--	--

Array vs Linked List ?

- Array memiliki ruang atau aksesibilitas yang terbatas. Sedangkan Linked bisa meng-Allokasi memori secara dinamis.

Linked List

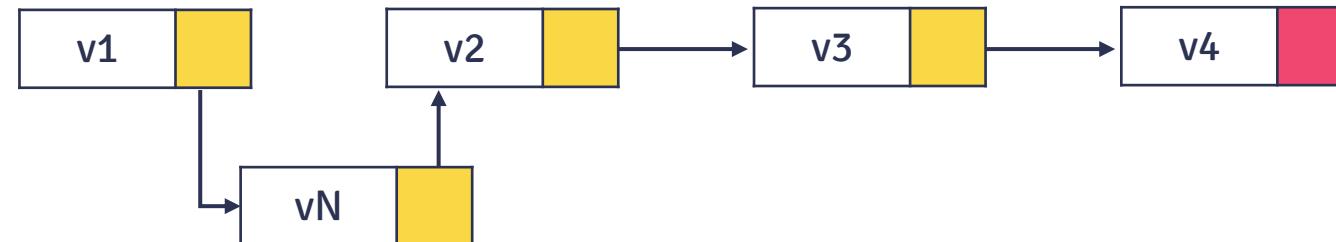


Tambahin valueN ke posisi ke-2

Linked List



Linked List

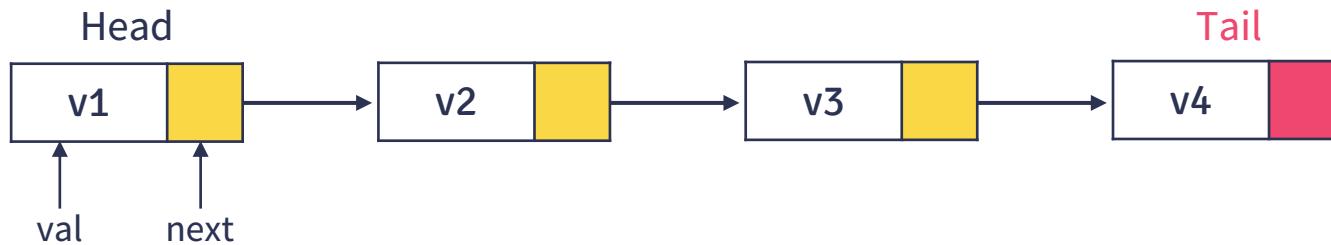


Tipe Linked List ?

- Single Linked List (Singly Linked List).
- Double Linked List (Doubly Linked List).
- Circular Linked List.
- Multiple Linked List.

Single Linked List ?

- Single Linked List merupakan suatu linked list yang hanya memiliki satu variabel pointer saja. Dimana pointer tersebut menunjuk ke node selanjutnya dan pointer pada tail menunjuk ke NULL.
- Navigasi item maju saja.
- Single Linked List terdiri dari sejumlah elemen (node) dimana setiap node memiliki penunjuk berikutnya ke elemen (node) berikutnya.
- Penunjuk node terakhir adalah NULL, yang menunjukkan akhir dari Single Linked List.



Deklarasi & Inisialisasi ?

```
/*
struct LinkListName{
    // komponen / member
    dataTypeData1 dataName1;
    ...
    LinkListName *next;
};
*/
```

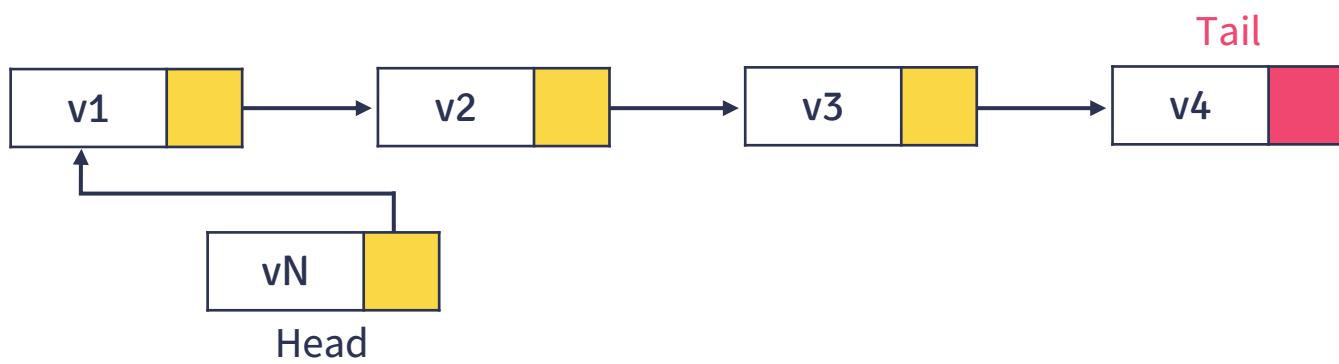
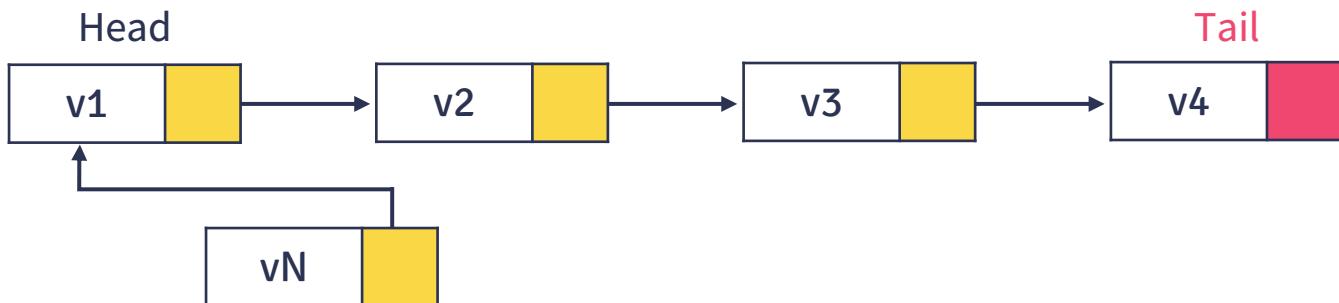
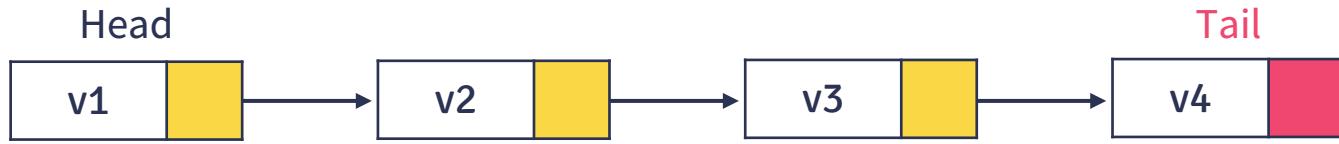
```
int main()
{
    /*
        LinkListName *node1, *nodeN;
        node1 = (LinkListName*) malloc(sizeof(LinkListName));
        nodeN = new LinkListName();
    */
    /*
        node1->dataName1 = valData1;
        ...
        node1->next = nodeN;

        nodeN->dataNameN = valDataN;
        ...
        nodeN->next = NULL;
    */
}
```

Print Single Linked List ?

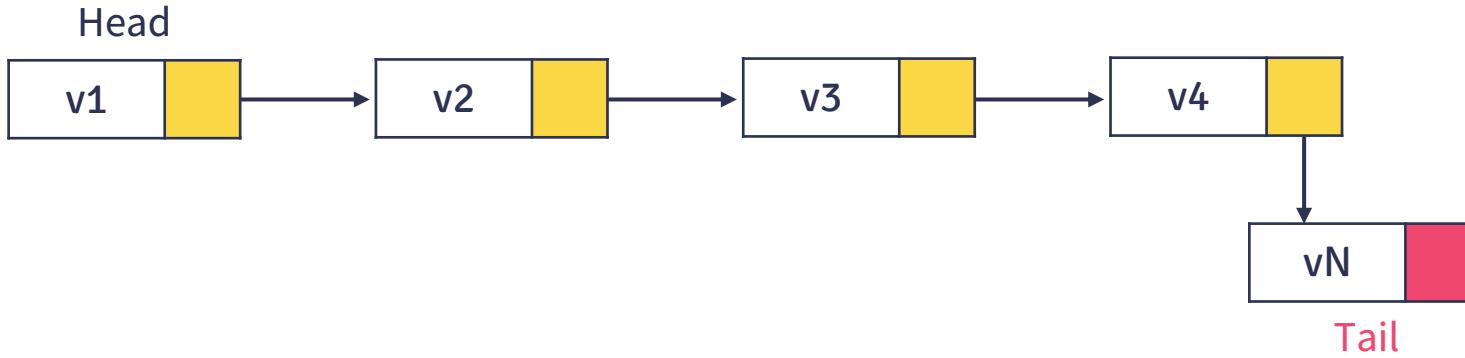
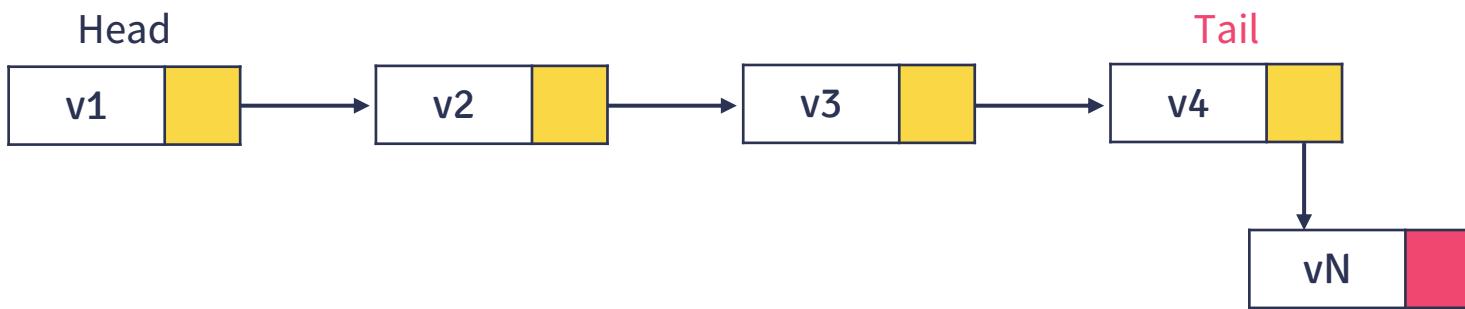
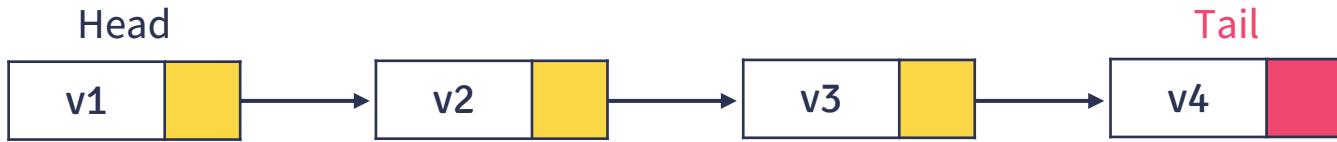
```
/*
LinkListName *cur;
cur = node1;
while( cur != NULL ){
    // print
    cur = cur->next;
}
*/
```

Added at Beginning Node ?

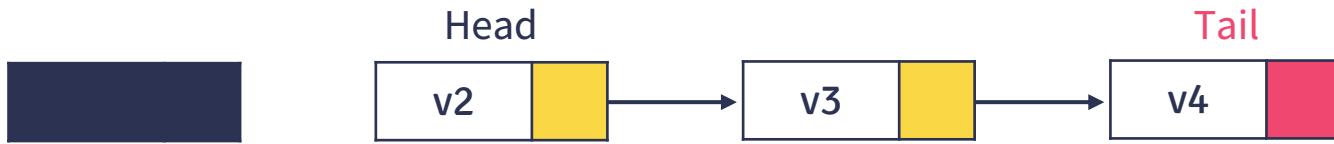
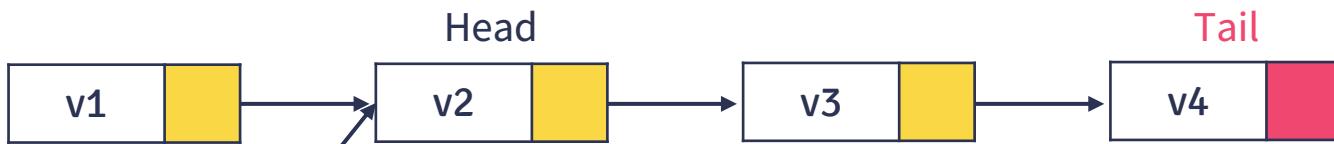
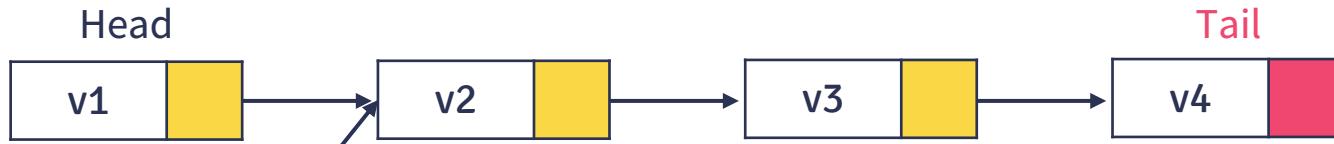


Head

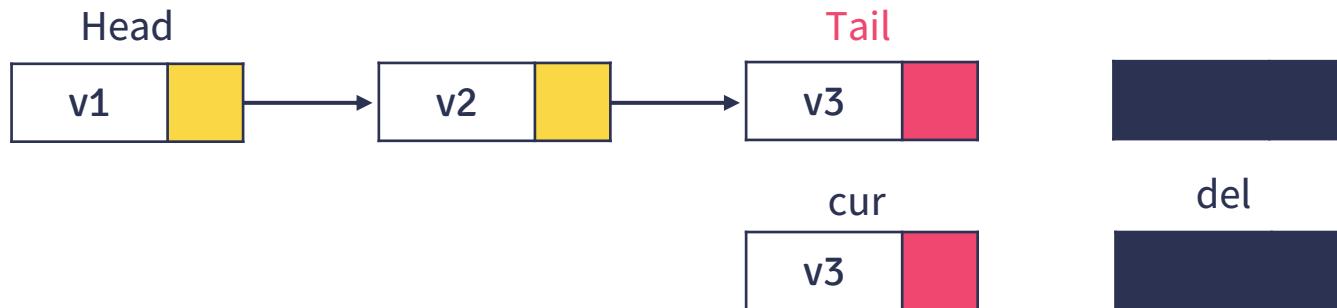
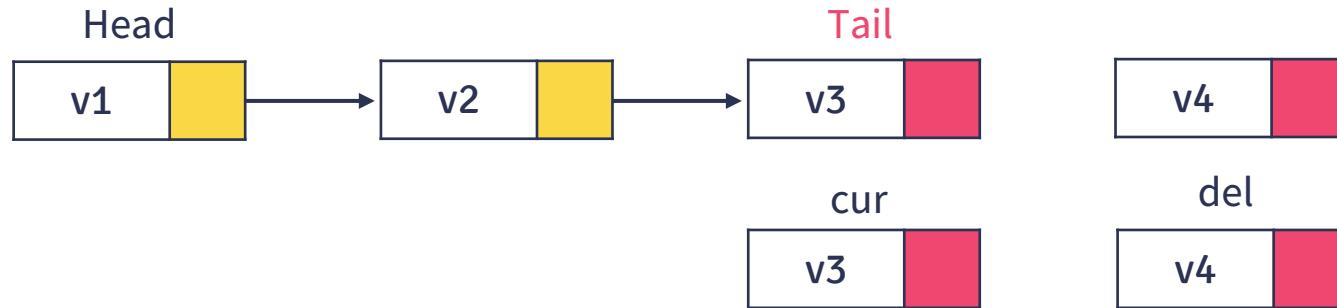
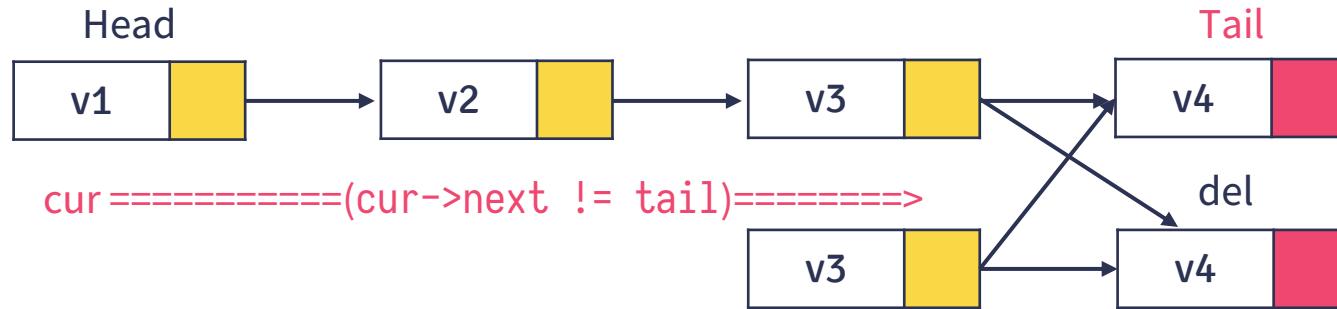
Added at Last Node ?



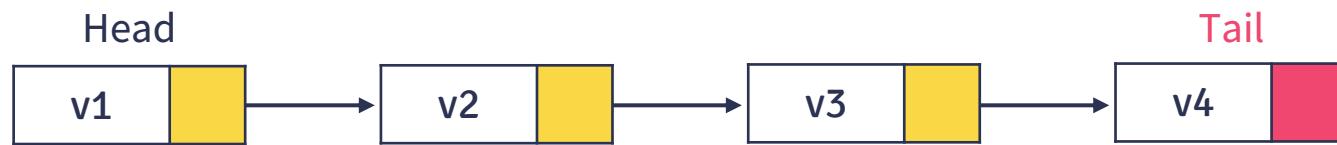
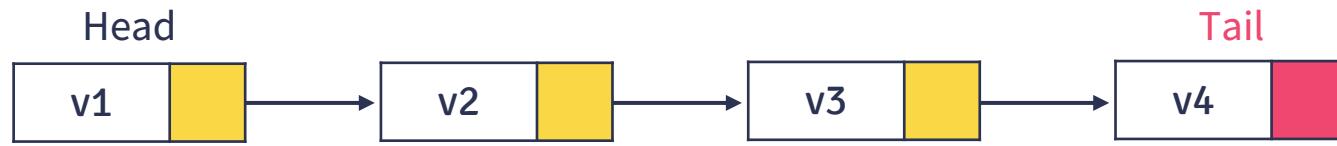
Delete the First Node ?



Delete the Last Node ?



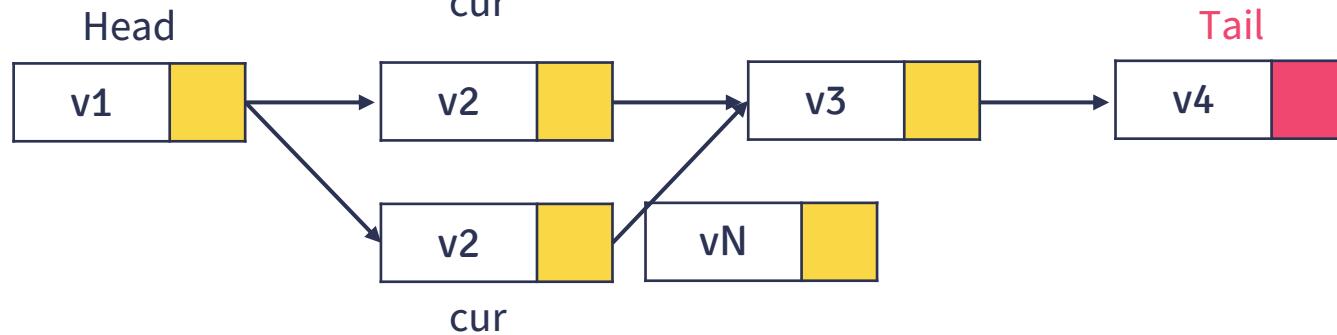
Added at Middle Node ?



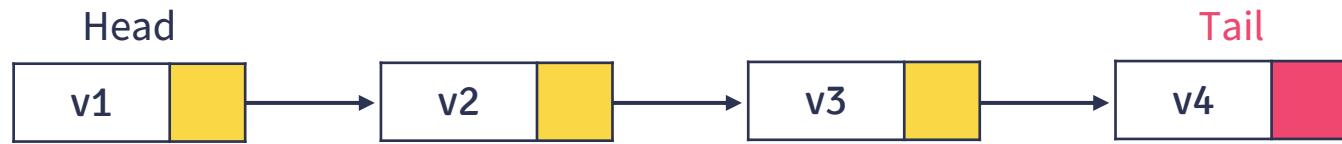
$\Rightarrow \text{loop } (1 < \text{posisi}-1) \Rightarrow$



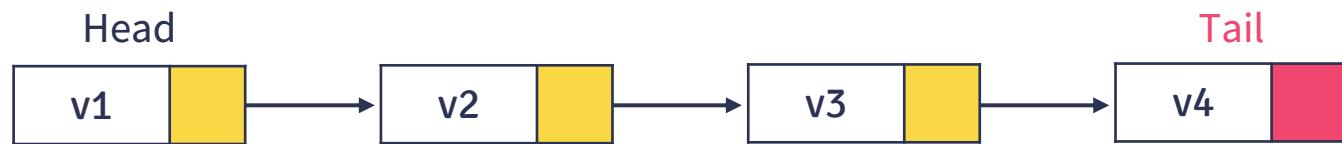
cur



Added at Middle Node ?



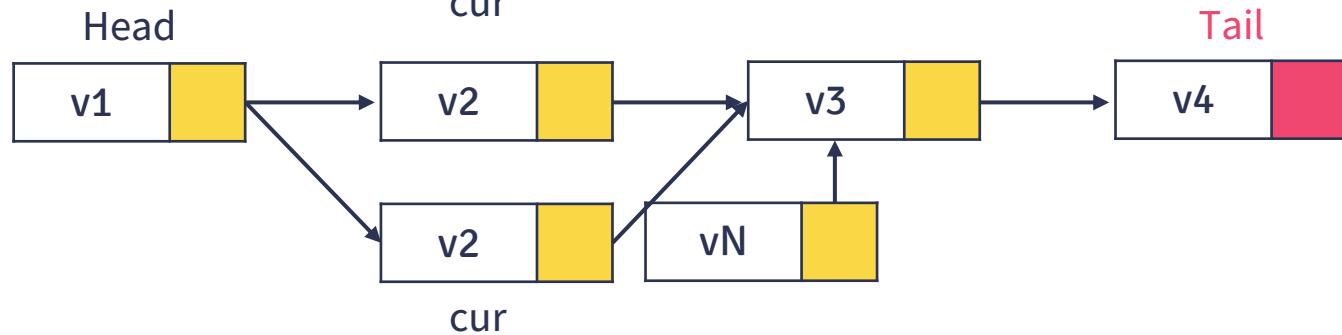
Tambah vN ke posisi 3



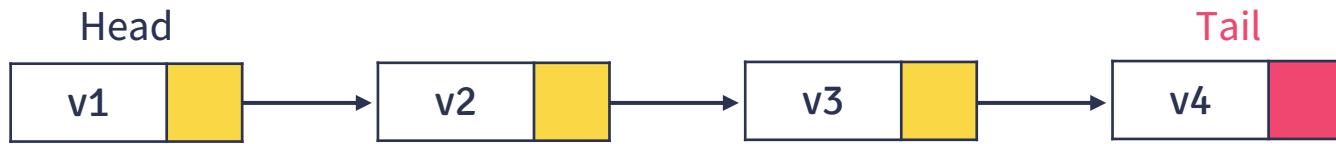
$\Rightarrow \text{loop } (1 < \text{posisi}-1) \Rightarrow$



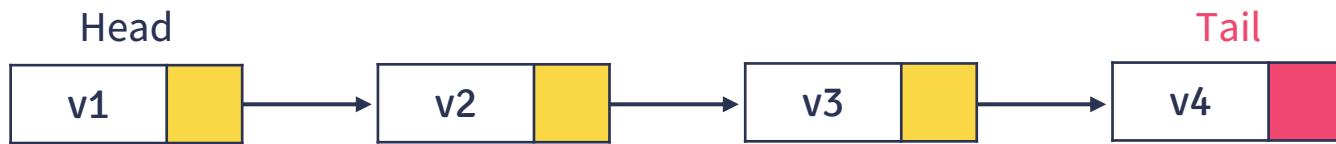
cur



Added at Middle Node ?



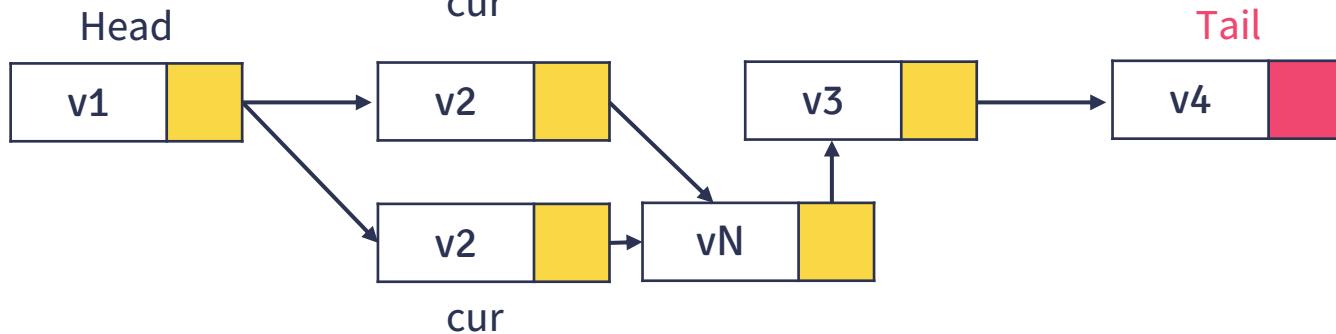
Tambah vN ke posisi 3



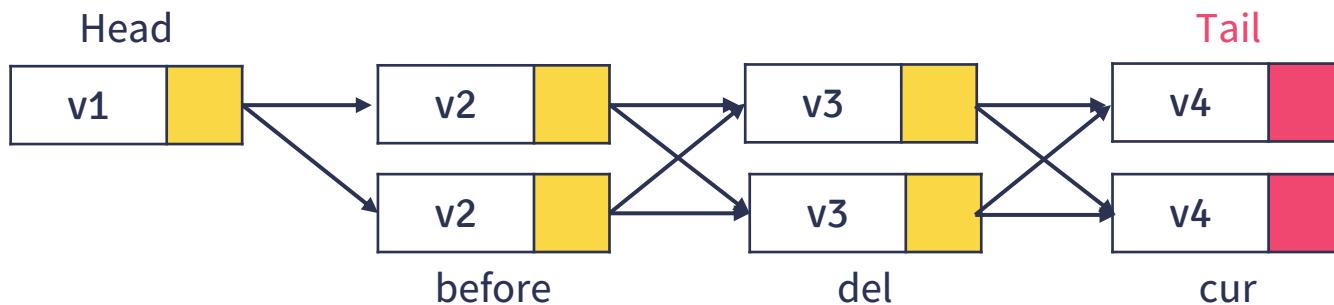
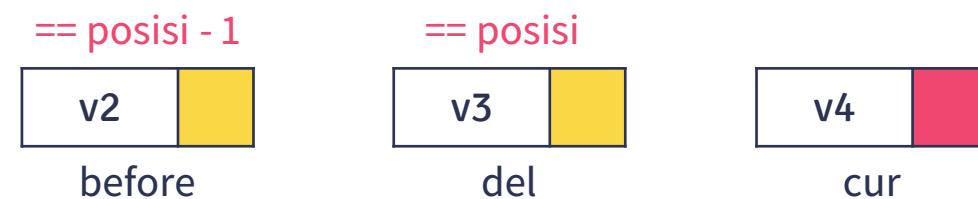
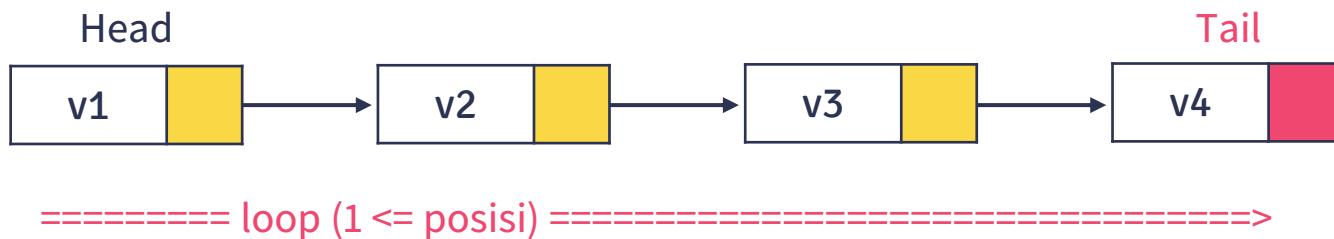
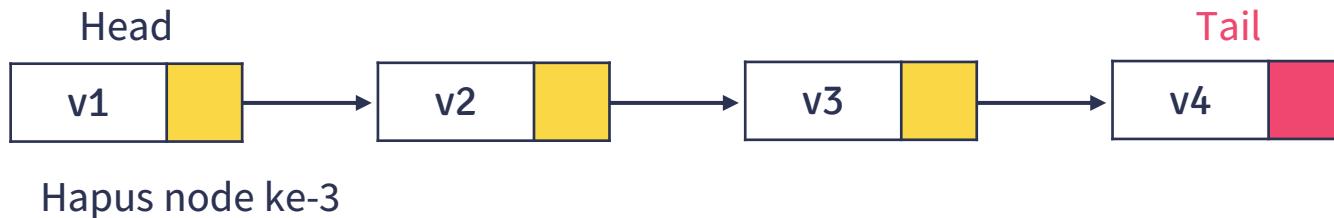
$\Rightarrow \text{loop } (1 < \text{posisi}-1) \Rightarrow$



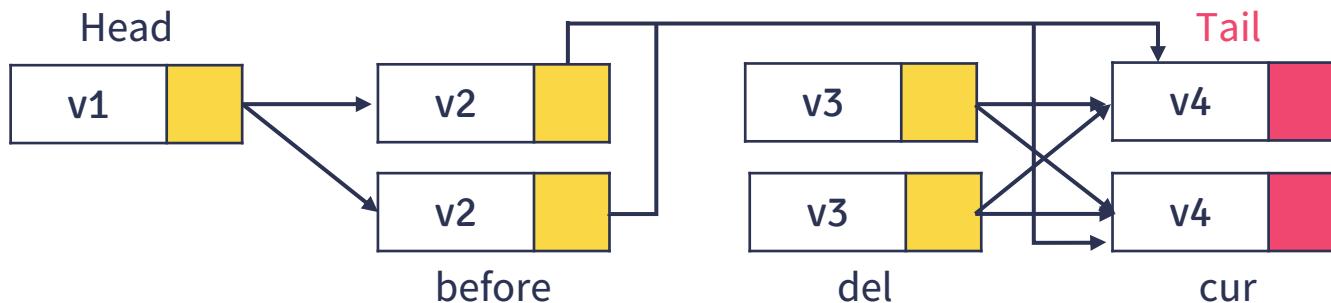
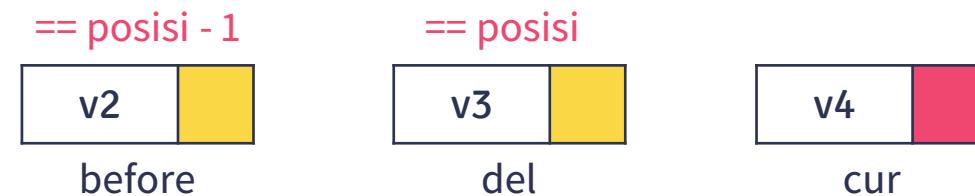
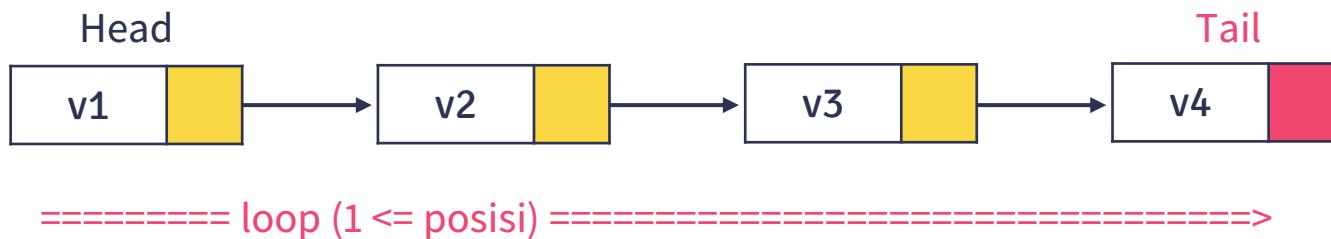
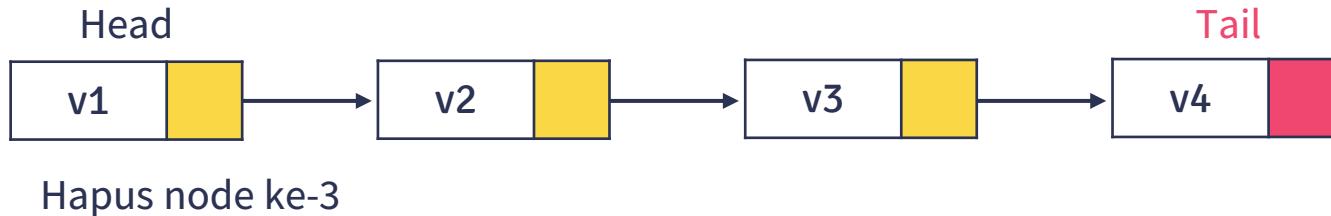
cur



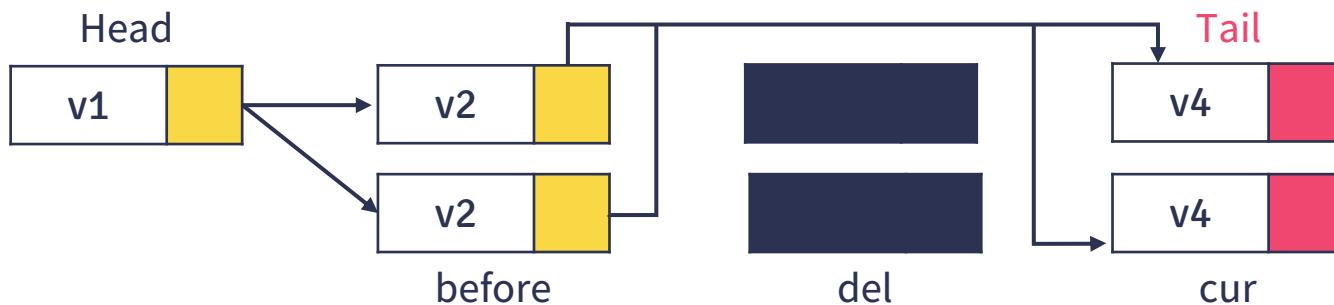
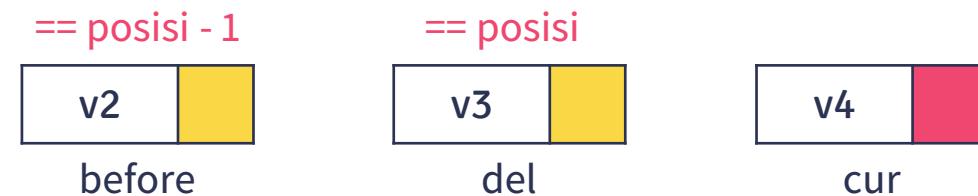
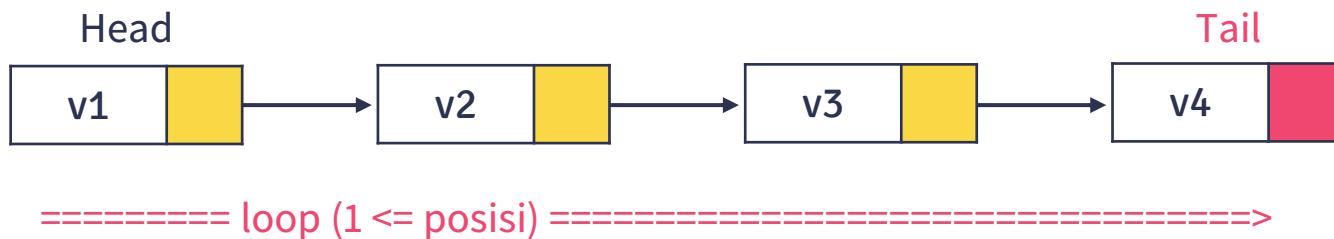
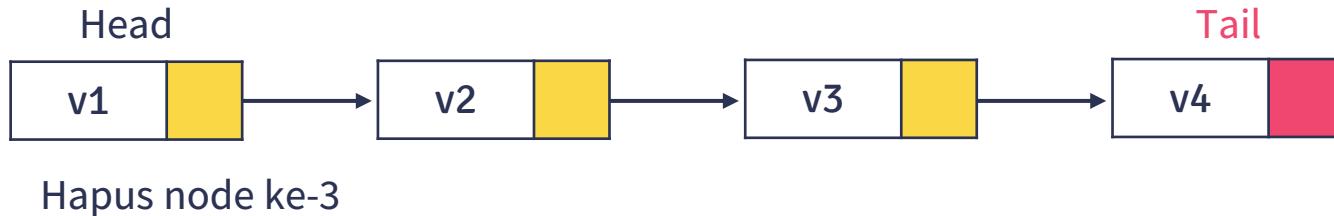
Delete at Middle Node ?



Delete at Middle Node ?



Delete at Middle Node ?





Video Selanjutnya

Double Linked List



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

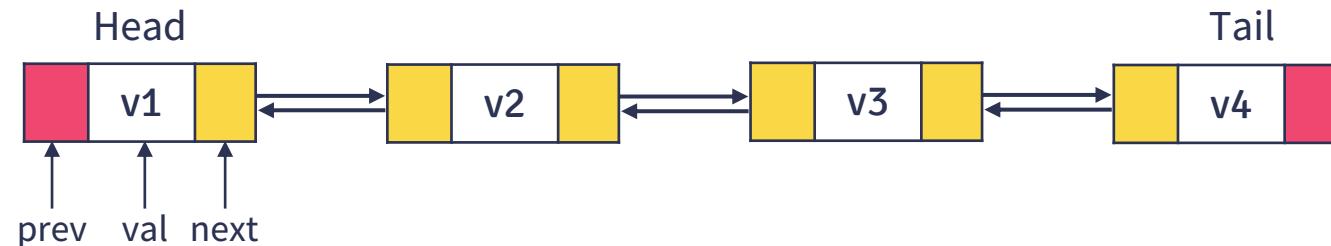
Saniati,S.ST.,M.T.

EPISODE 4B

Double Linked List

Double Linked List ?

- Double Linked List merupakan suatu linked list yang memiliki dua variabel pointer. Dimana pointer tersebut menunjuk ke node sebelum dan selanjutnya (prev & next).
- Double Linked List terdiri dari sejumlah elemen (node) dimana setiap node memiliki penunjuk prev (menunjuk node sebelumnya) dan next (menunjuk node selanjutnya).
- Penunjuk prev pada node head menunjuk ke NULL, menandakan bahwa node head (node awal).
- Penunjuk next pada node tail menunjuk ke NULL, menandakan bahwa node tail (node akhir).



Deklarasi & Inisialisasi ?

```
/*
struct LinkListName{
    // komponen / member
    dataTypeData1 dataName1;
    . . .
    LinkListName *prev;
    LinkListName *next;
};
*/
```

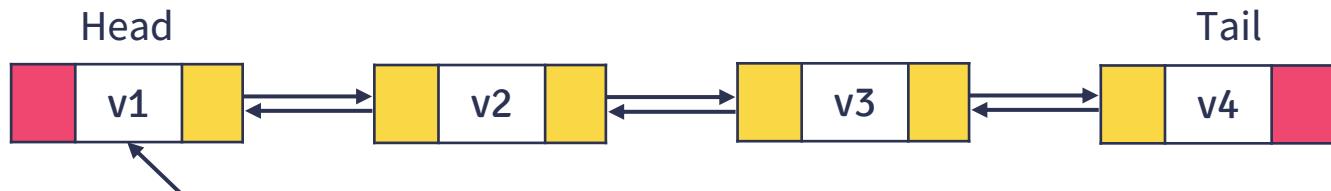
```
/*
LinkListName *head, *tail;
head = (LinkListName*) malloc(sizeof(LinkListName));
tail = new LinkListName();
*/
/*
head->dataName1 = valData1;
. . .
head->prev = NULL;
head->next = tail;

tail->dataName1 = valData1;
. . .
tail->prev = head;
tail->next = NULL;
*/
*/
```

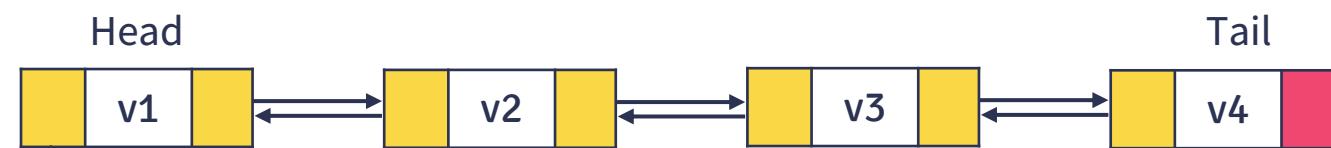
Print Double Linked List ?

```
/*
LinkListName *cur;
cur = head;
while( cur != NULL ){
    // print
    cur = cur->next;
}
*/
```

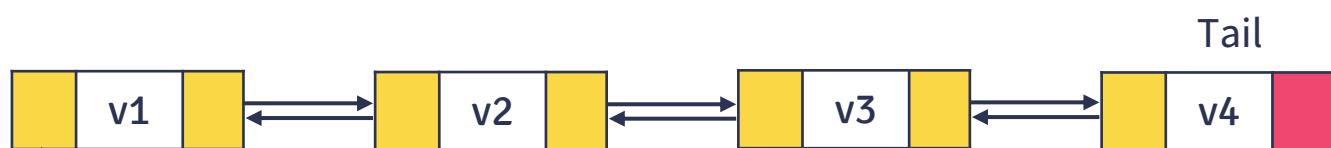
Added at Beginning Node ?



vN
newNode

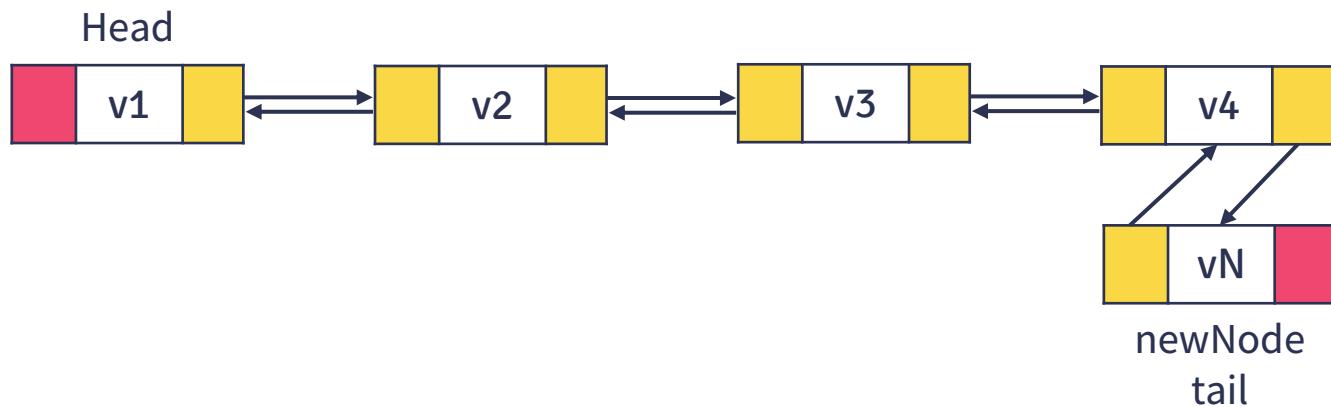
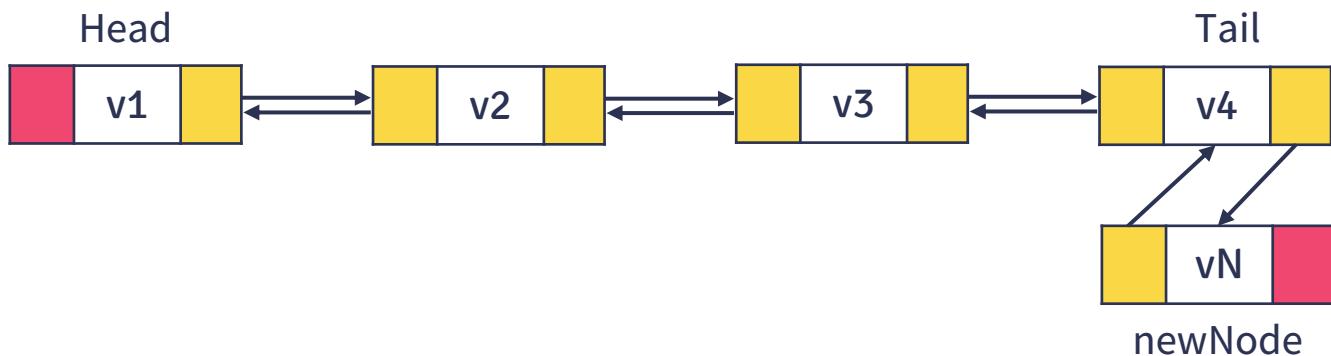
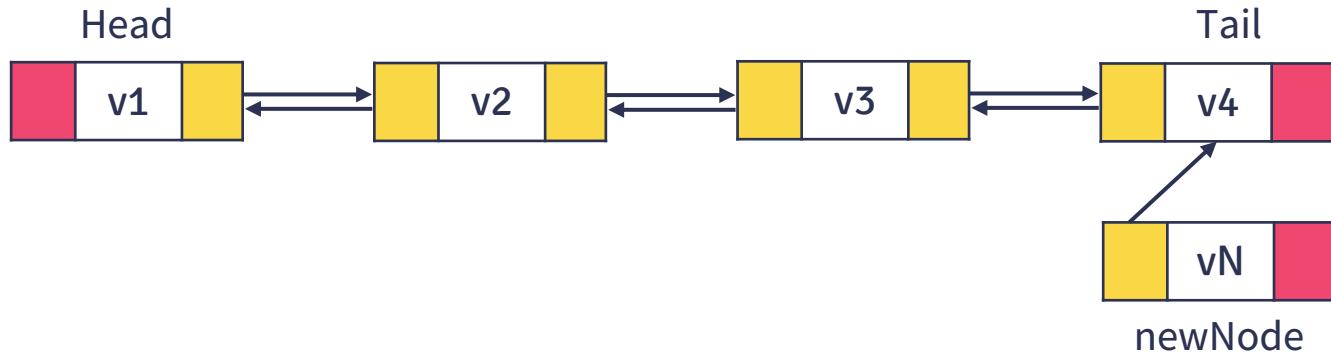


vN
newNode

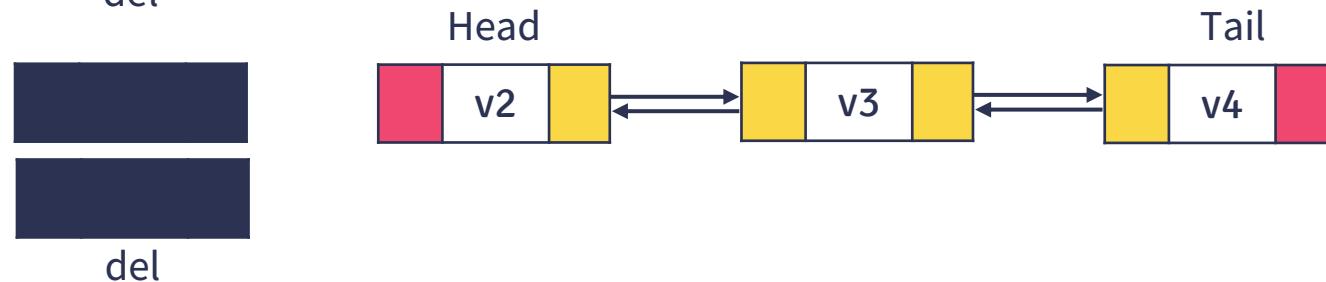
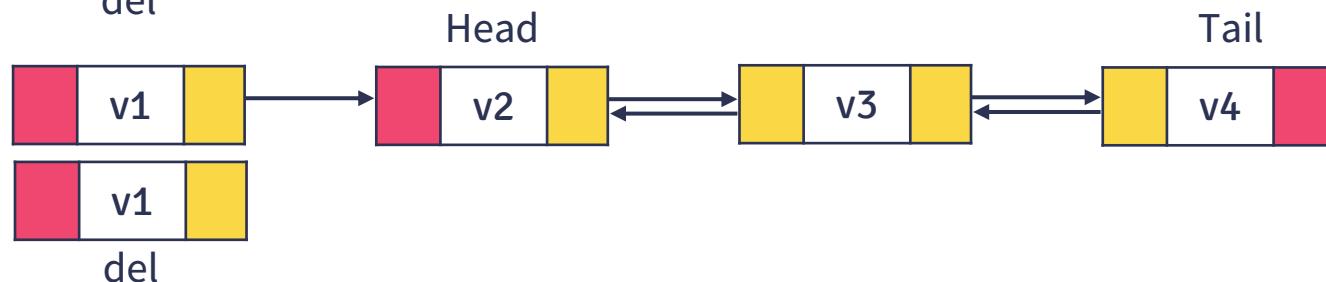
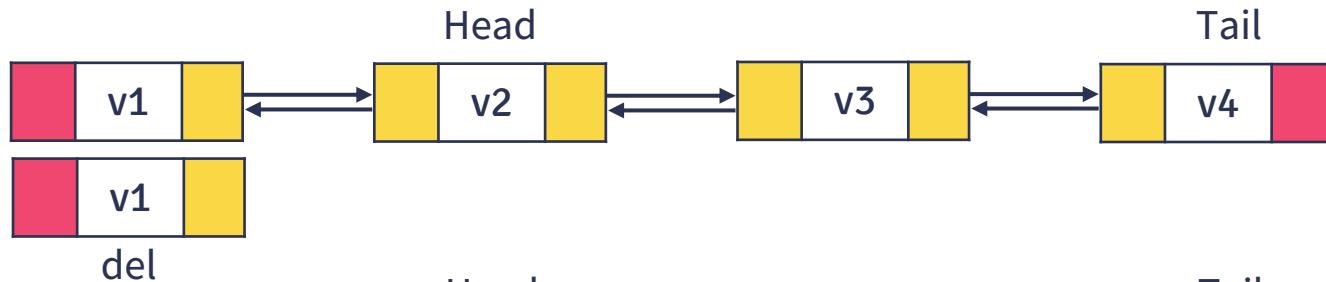
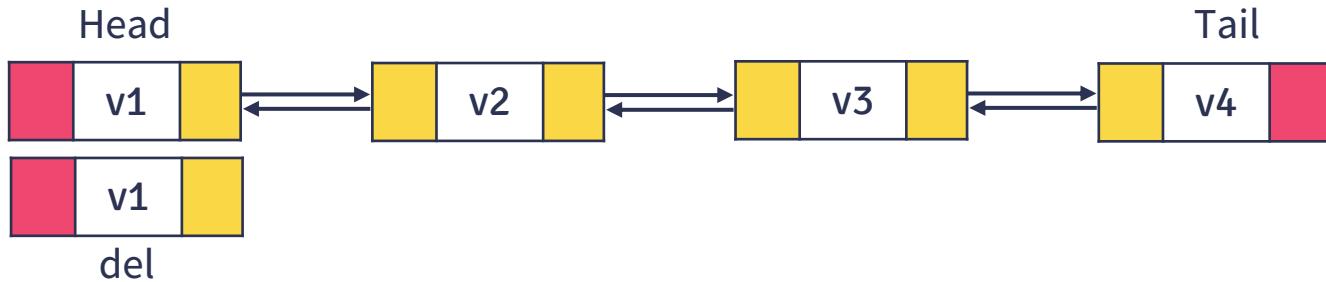


vN
newNode
head

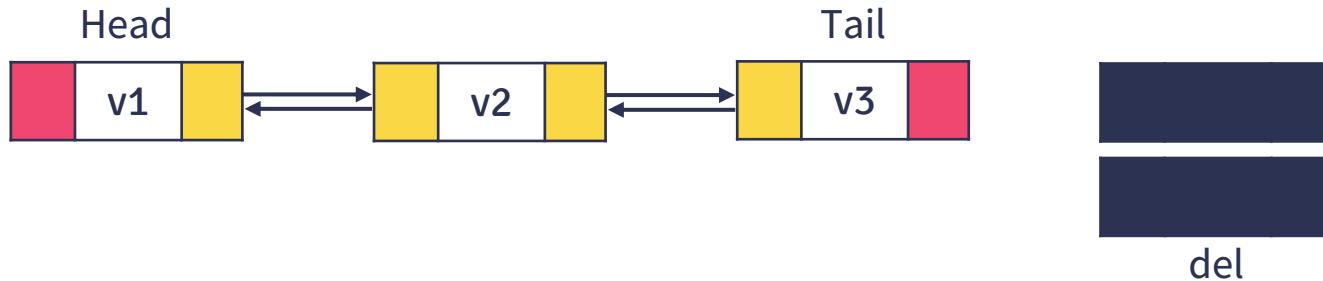
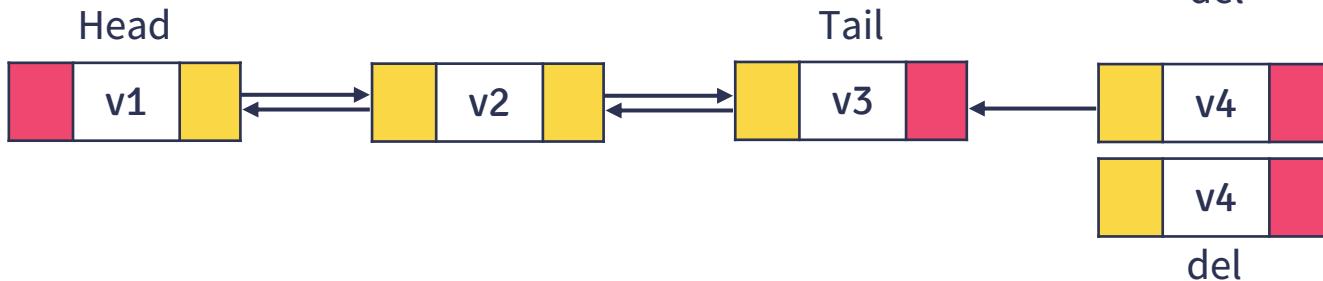
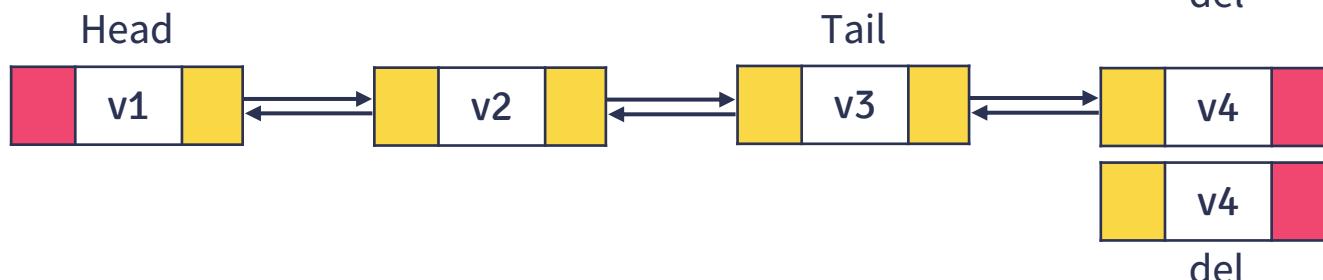
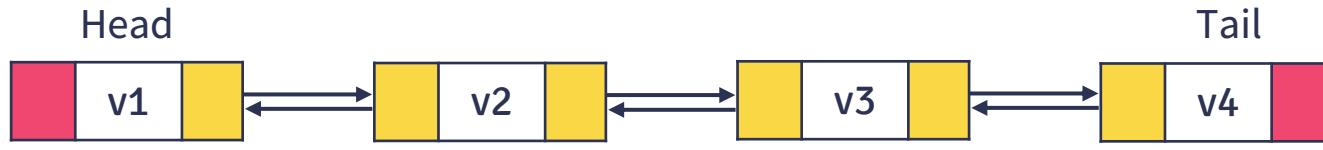
Added at Last Node ?



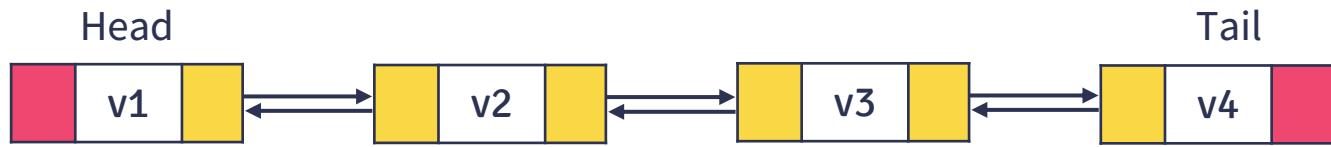
Delete the First Node ?



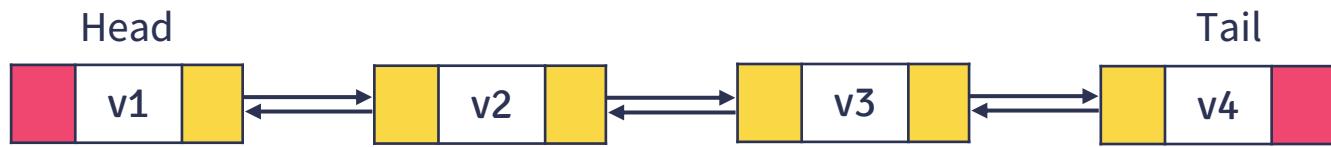
Delete the Last Node ?



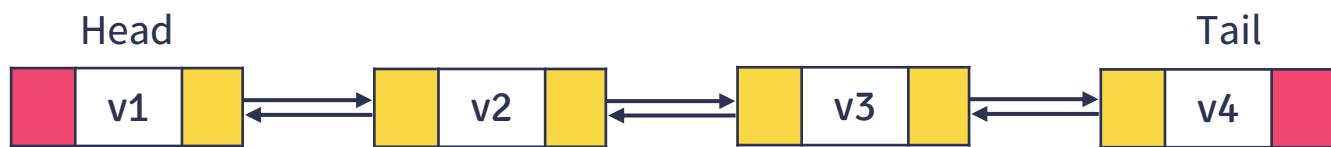
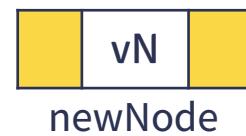
Added at Middle Node ?



Tambah vN ke posisi 3

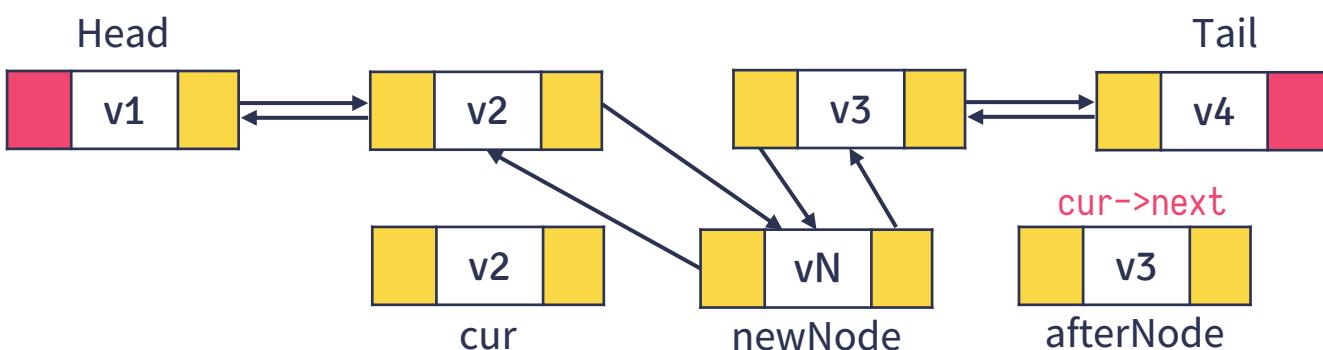
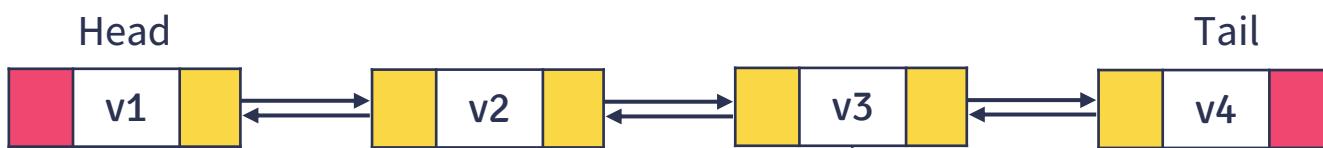
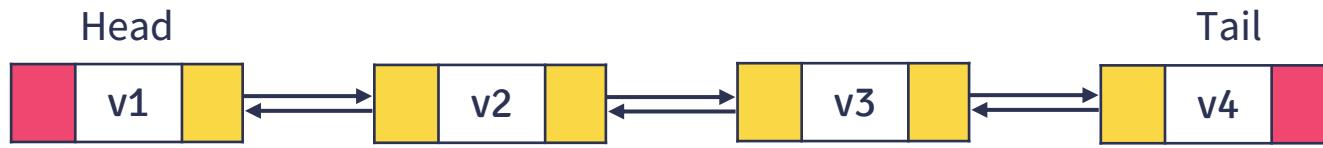


cur == (1 < posisi - 1) ==>

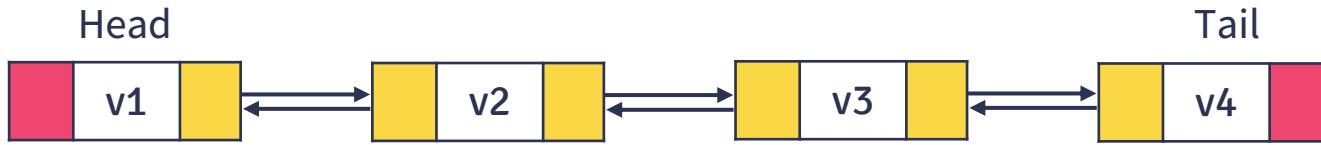


cur->next

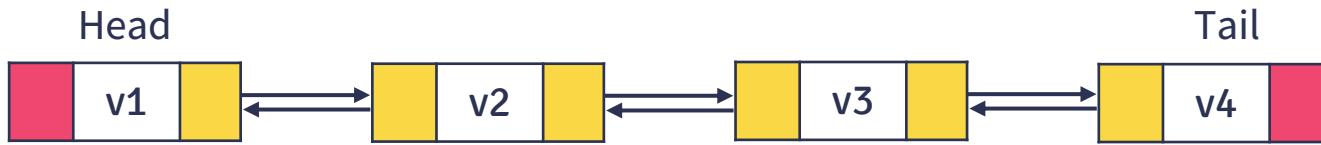
Added at Middle Node ?



Delete at Middle Node ?



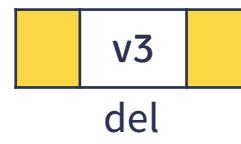
Hapus posisi ke-3



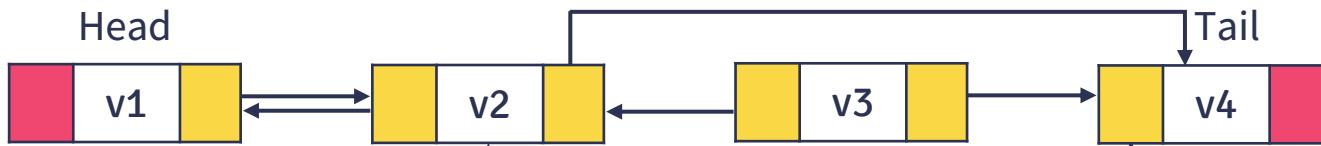
$\text{cur} == (\text{1} < \text{posisi} - 1) ==>$



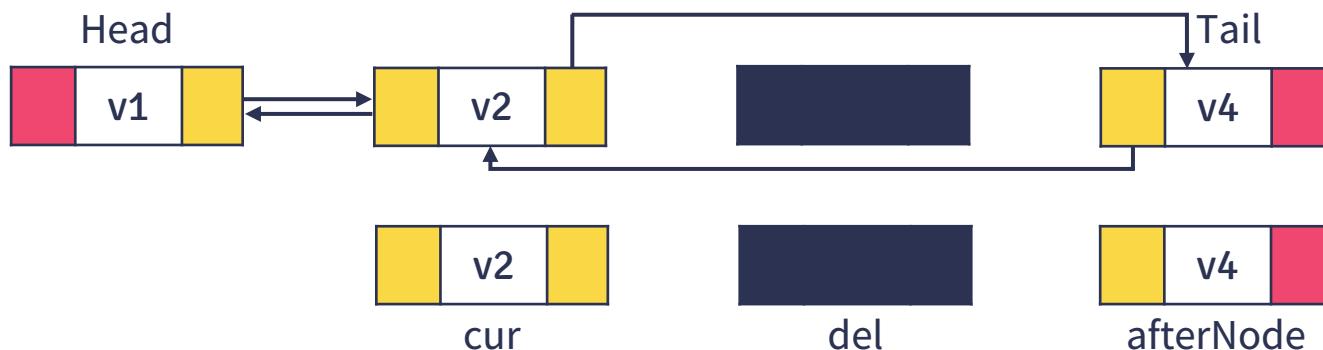
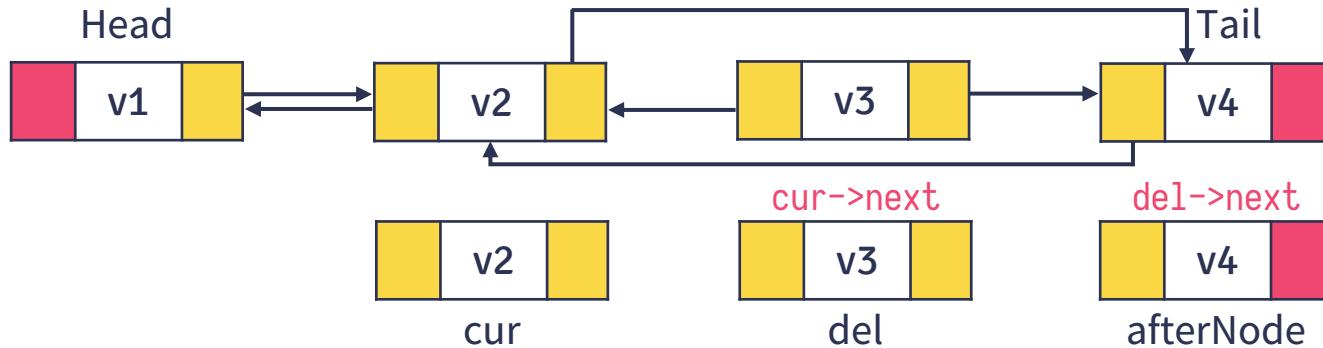
$\text{cur} \rightarrow \text{next}$



$\text{del} \rightarrow \text{next}$



Delete at Middle Node ?





Video Selanjutnya

Circular Single Linked List



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

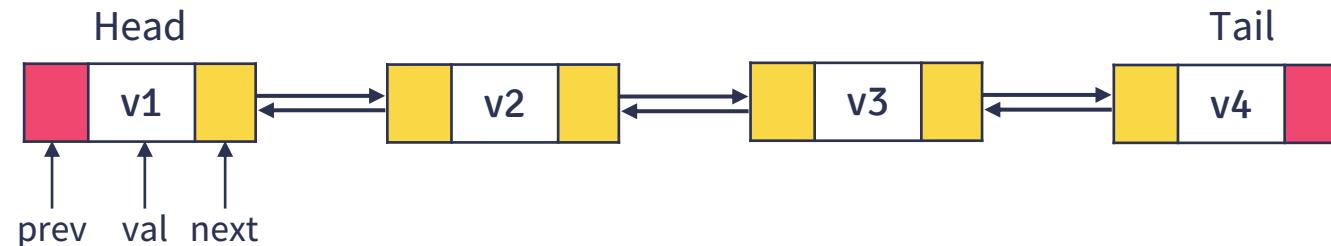
Saniati,S.ST.,M.T.

EPISODE 4B

Double Linked List

Double Linked List ?

- Double Linked List merupakan suatu linked list yang memiliki dua variabel pointer. Dimana pointer tersebut menunjuk ke node sebelum dan selanjutnya (prev & next).
- Double Linked List terdiri dari sejumlah elemen (node) dimana setiap node memiliki penunjuk prev (menunjuk node sebelumnya) dan next (menunjuk node selanjutnya).
- Penunjuk prev pada node head menunjuk ke NULL, menandakan bahwa node head (node awal).
- Penunjuk next pada node tail menunjuk ke NULL, menandakan bahwa node tail (node akhir).



Deklarasi & Inisialisasi ?

```
/*
struct LinkListName{
    // komponen / member
    dataTypeData1 dataName1;
    . . .
    LinkListName *prev;
    LinkListName *next;
};
*/
```

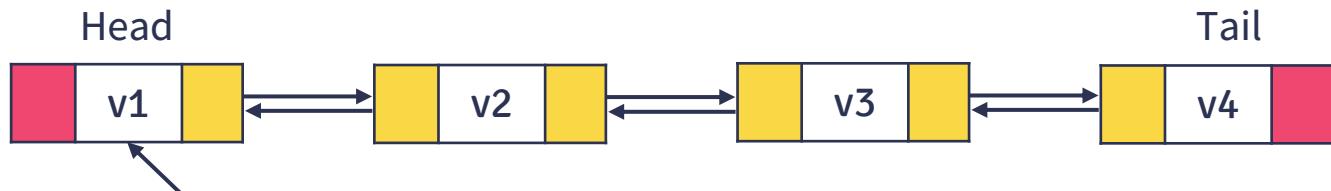
```
/*
LinkListName *head, *tail;
head = (LinkListName*) malloc(sizeof(LinkListName));
tail = new LinkListName();
*/
/*
head->dataName1 = valData1;
. . .
head->prev = NULL;
head->next = tail;

tail->dataName1 = valData1;
. . .
tail->prev = head;
tail->next = NULL;
*/
*/
```

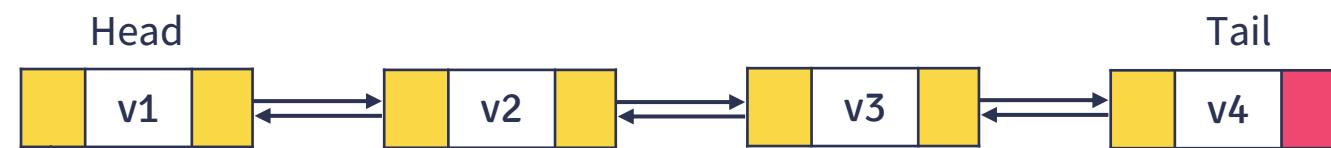
Print Double Linked List ?

```
/*
LinkListName *cur;
cur = head;
while( cur != NULL ){
    // print
    cur = cur->next;
}
*/
```

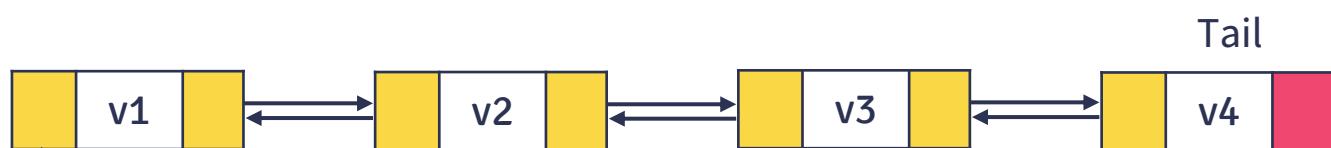
Added at Beginning Node ?



vN
newNode

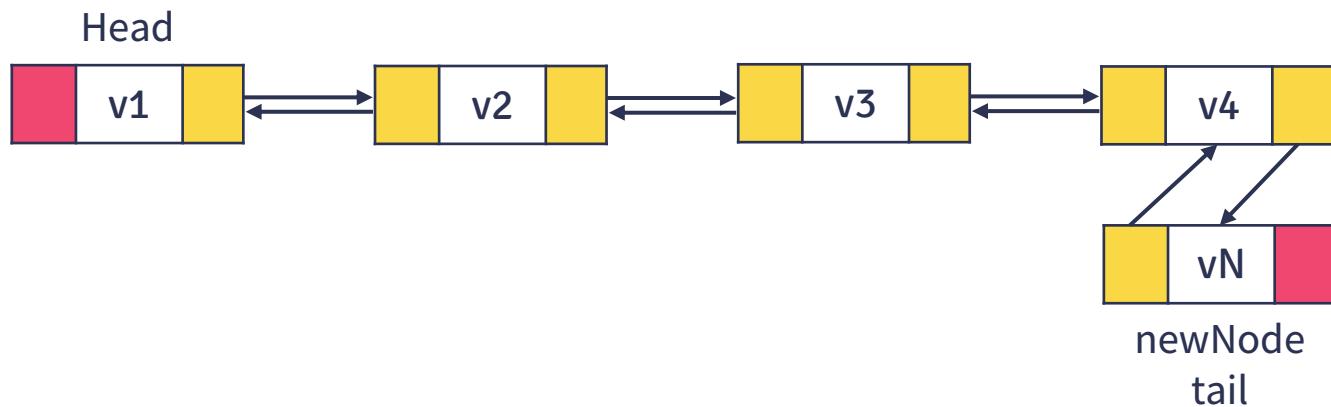
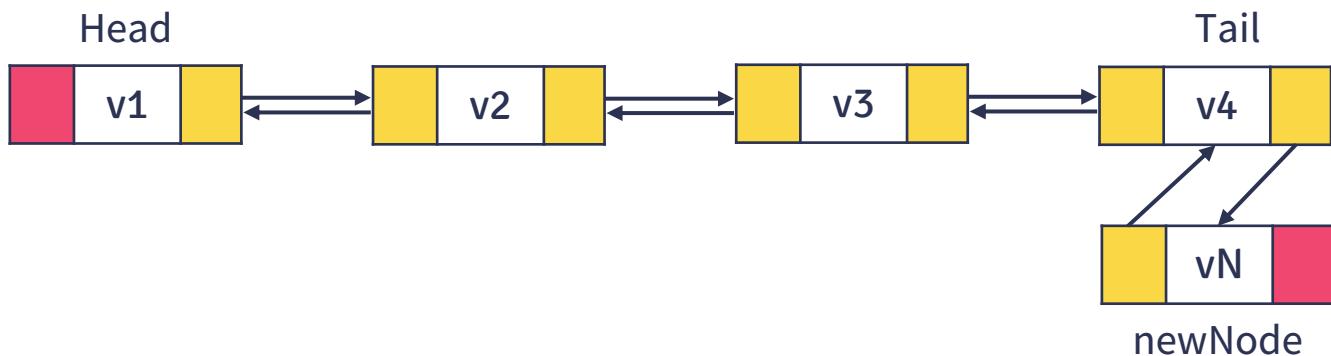
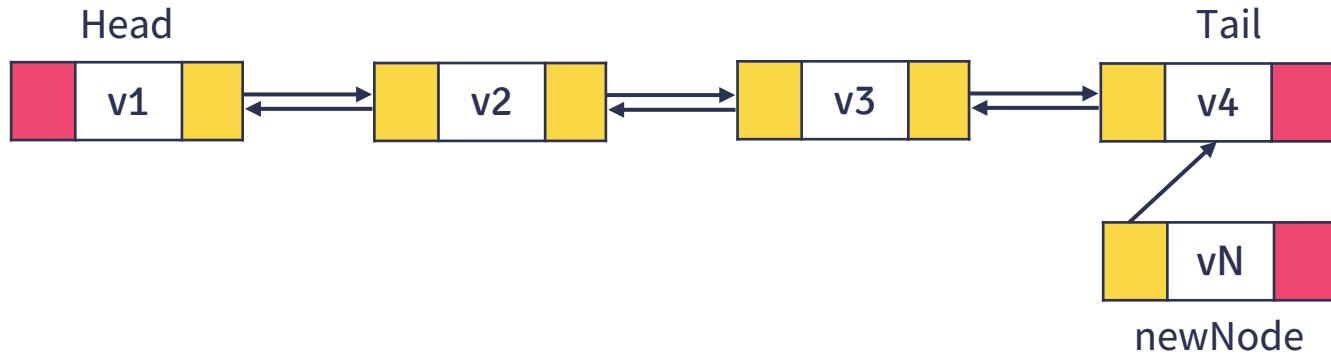


vN
newNode

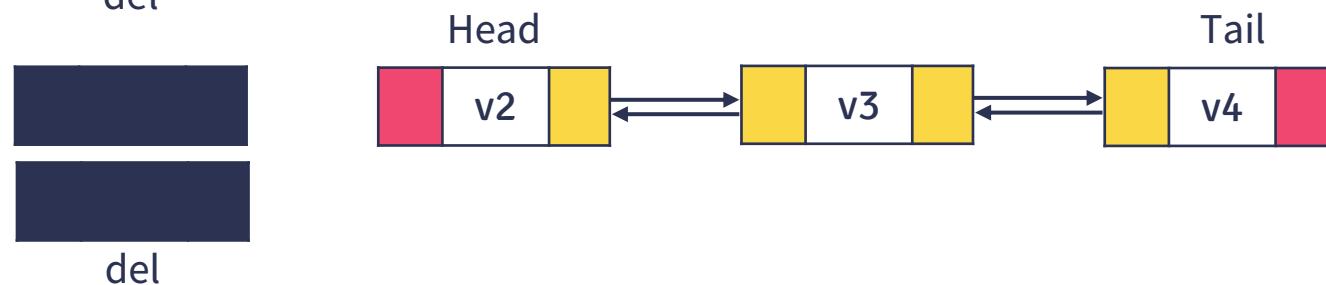
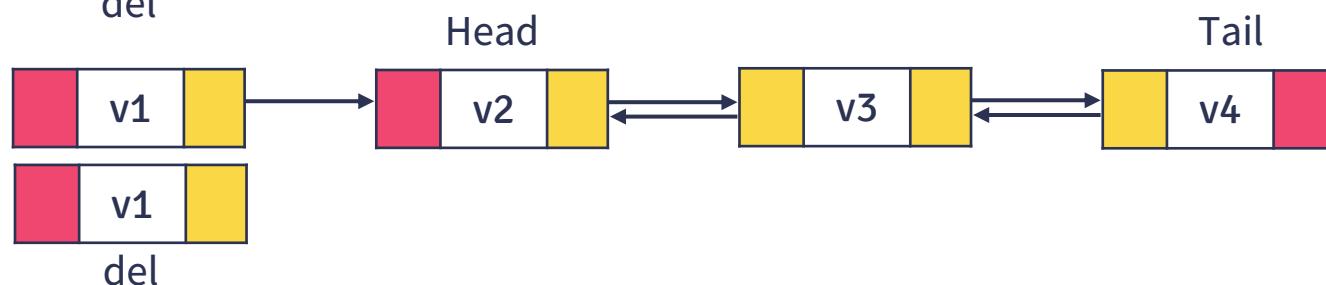
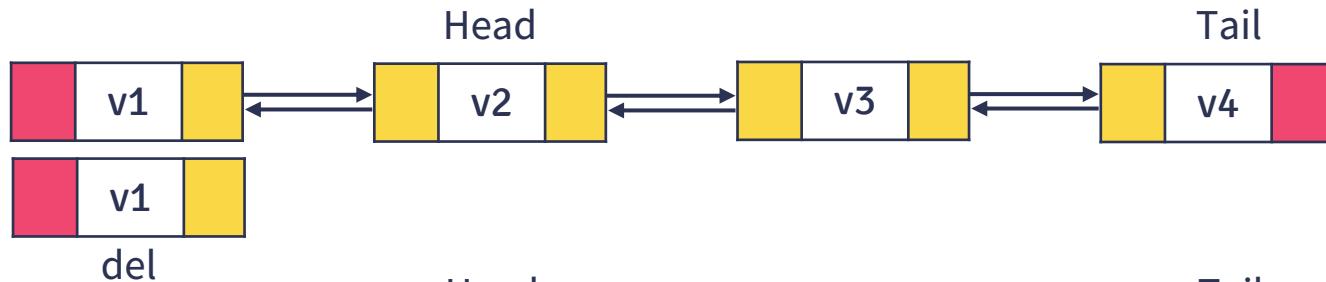
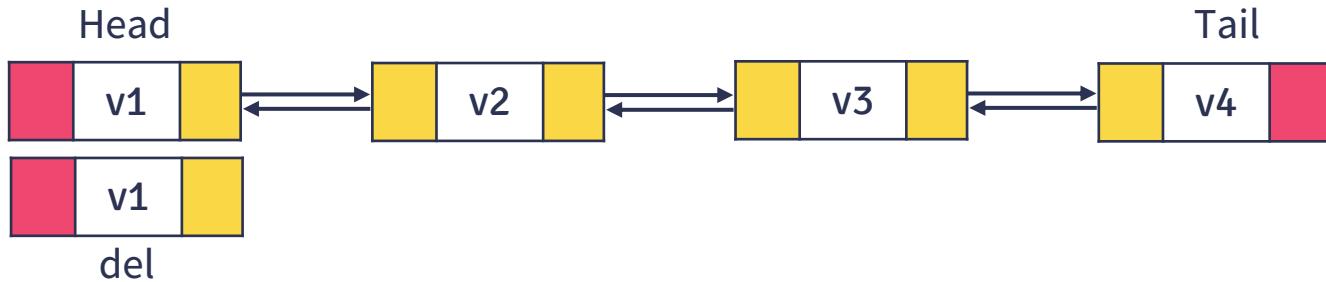


vN
newNode
head

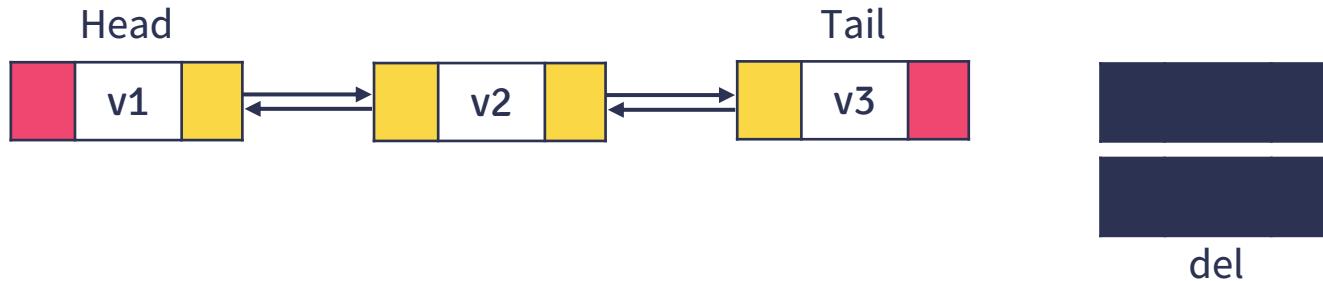
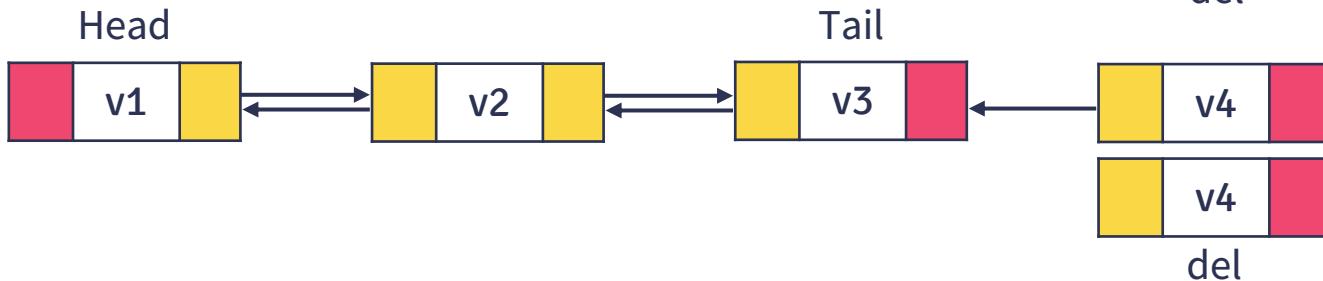
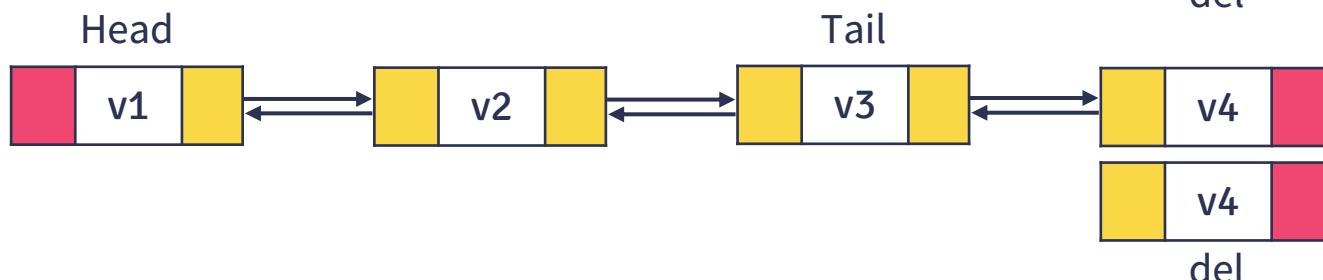
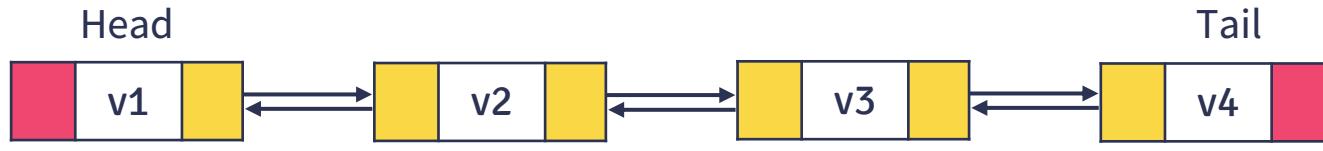
Added at Last Node ?



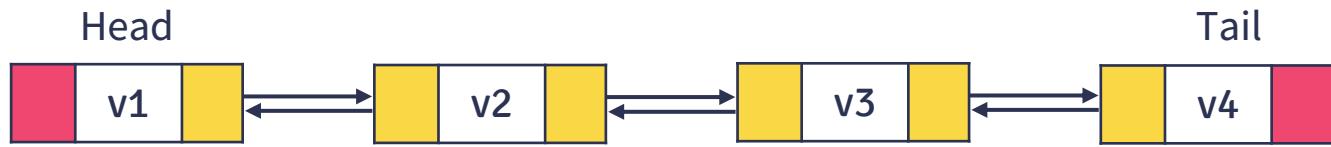
Delete the First Node ?



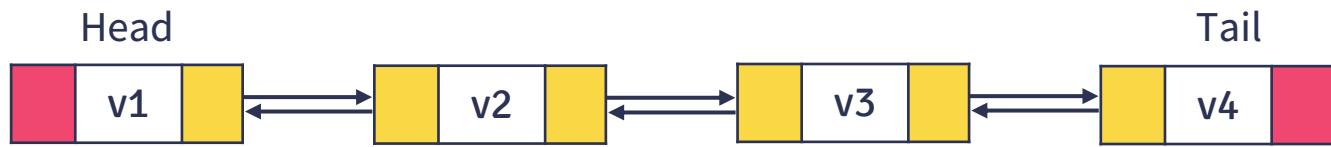
Delete the Last Node ?



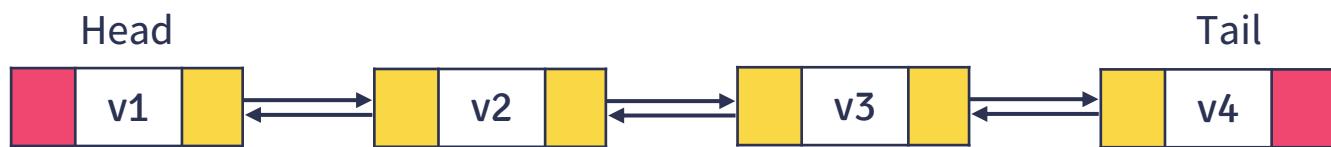
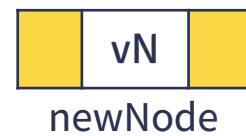
Added at Middle Node ?



Tambah vN ke posisi 3

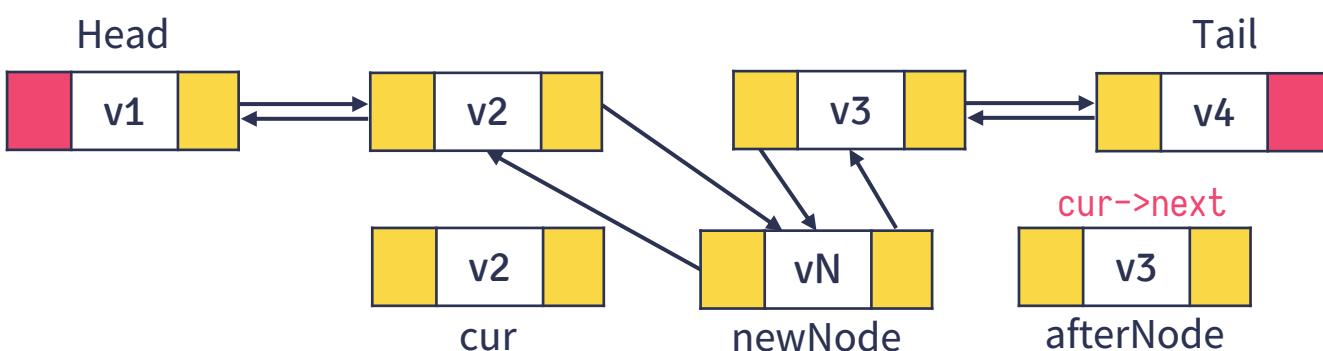
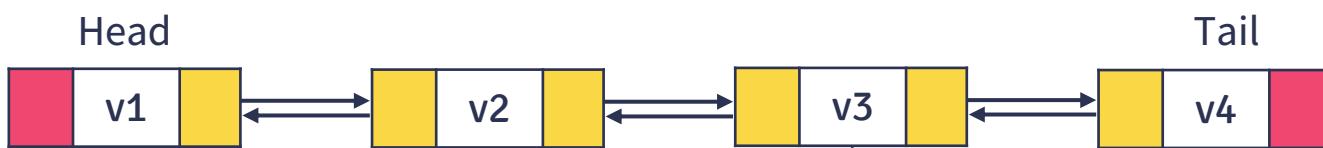
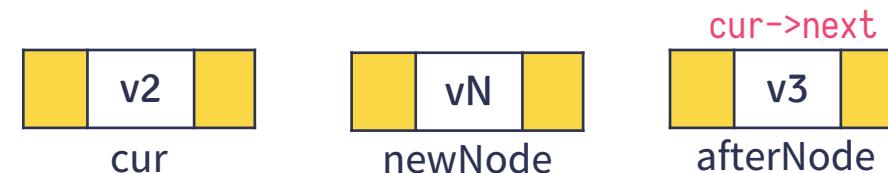
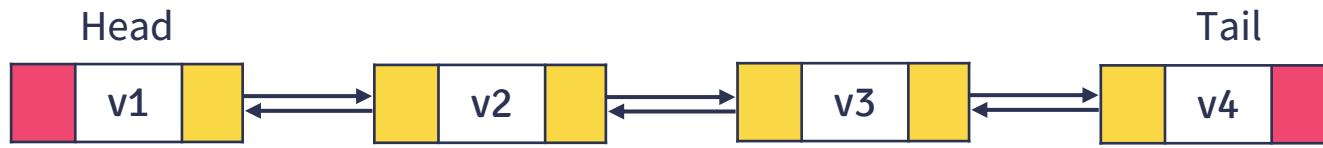


cur == (1 < posisi - 1) ==>

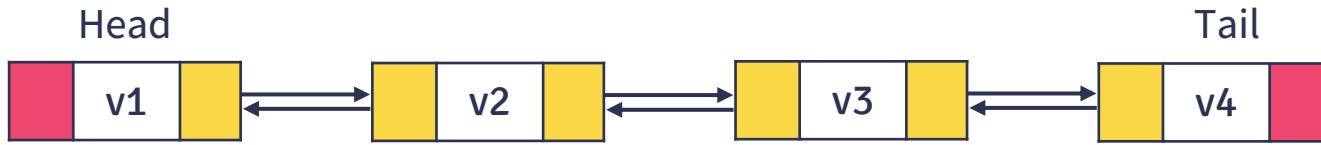


cur->next

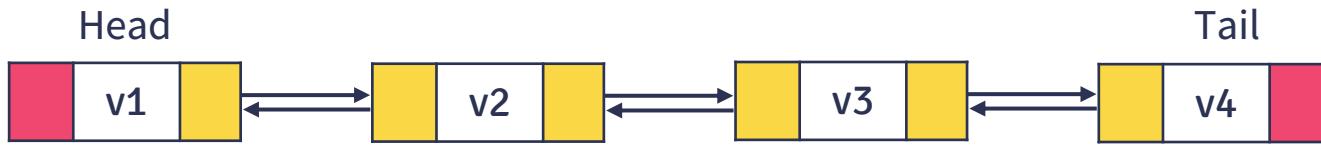
Added at Middle Node ?



Delete at Middle Node ?



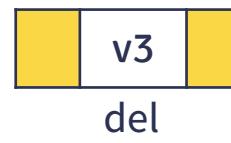
Hapus posisi ke-3



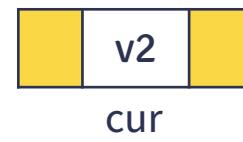
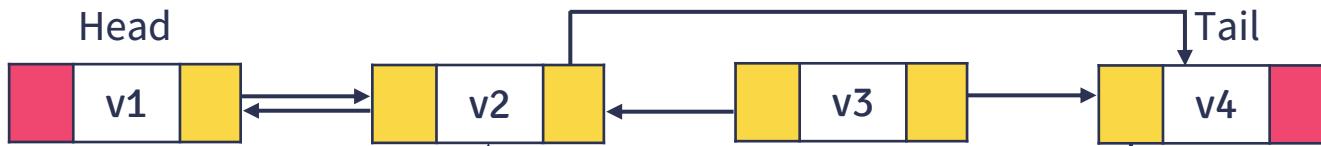
$\text{cur} == (\text{1} < \text{posisi} - 1) ==>$



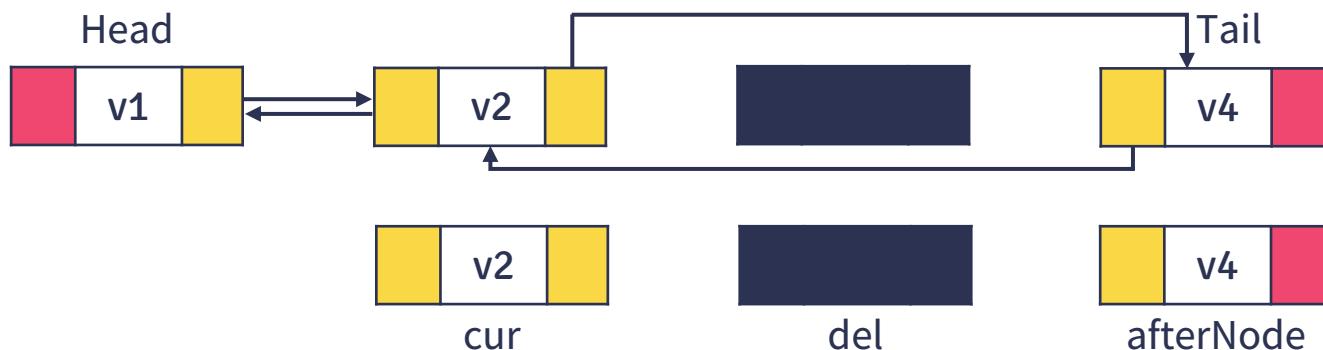
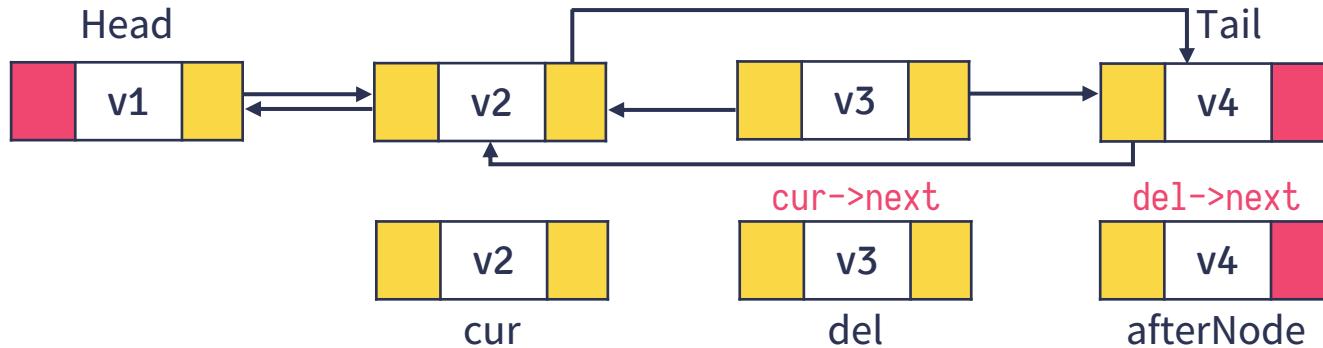
$\text{cur} \rightarrow \text{next}$



$\text{del} \rightarrow \text{next}$



Delete at Middle Node ?





Video Selanjutnya

Circular Single Linked List



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

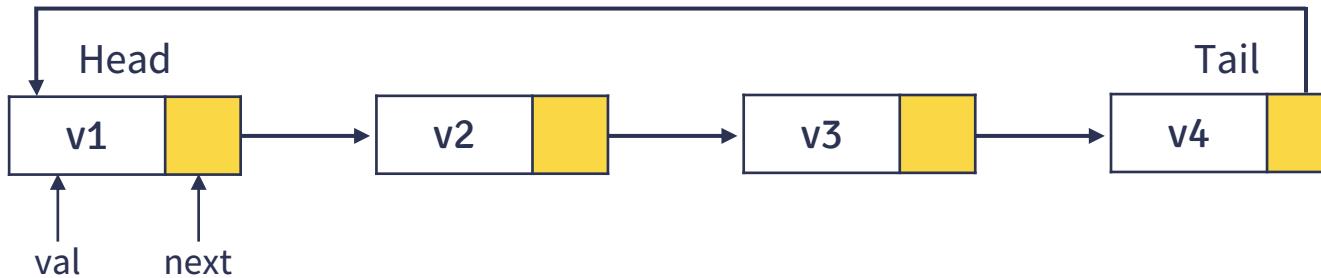
Saniati,S.ST.,M.T.

EPISODE **4C**

Circular Single Linked List

Circular Single Linked List ?

- Circular Linked List adalah sekumpulan node atau simpul yang tidak terdapat nilai NULL pada satupun nodenya.
- Bentuk node pada Circular Single Linked List sama dengan Single Linked List yang mempunyai data dan pointer next.
- Yang membedakan adalah jika pada Single Linked List node terakhir menunjuk ke NULL, tapi di Circular Single Linked List node terakhir menunjuk ke node pertama.



Deklarasi & Inisialisasi ?

```
// Deklarasi
/*
struct LinkedListName{
    typeData dataName1;
    . .
    LinkedListName *next;
};*/
*/
```

```
// Inisialisasi
/*
LinkedListName *head, *tail;
head = new LinkedListName();
tail = new LinkedListName();

head->dataName1 = val;
. .
head->next = tail;

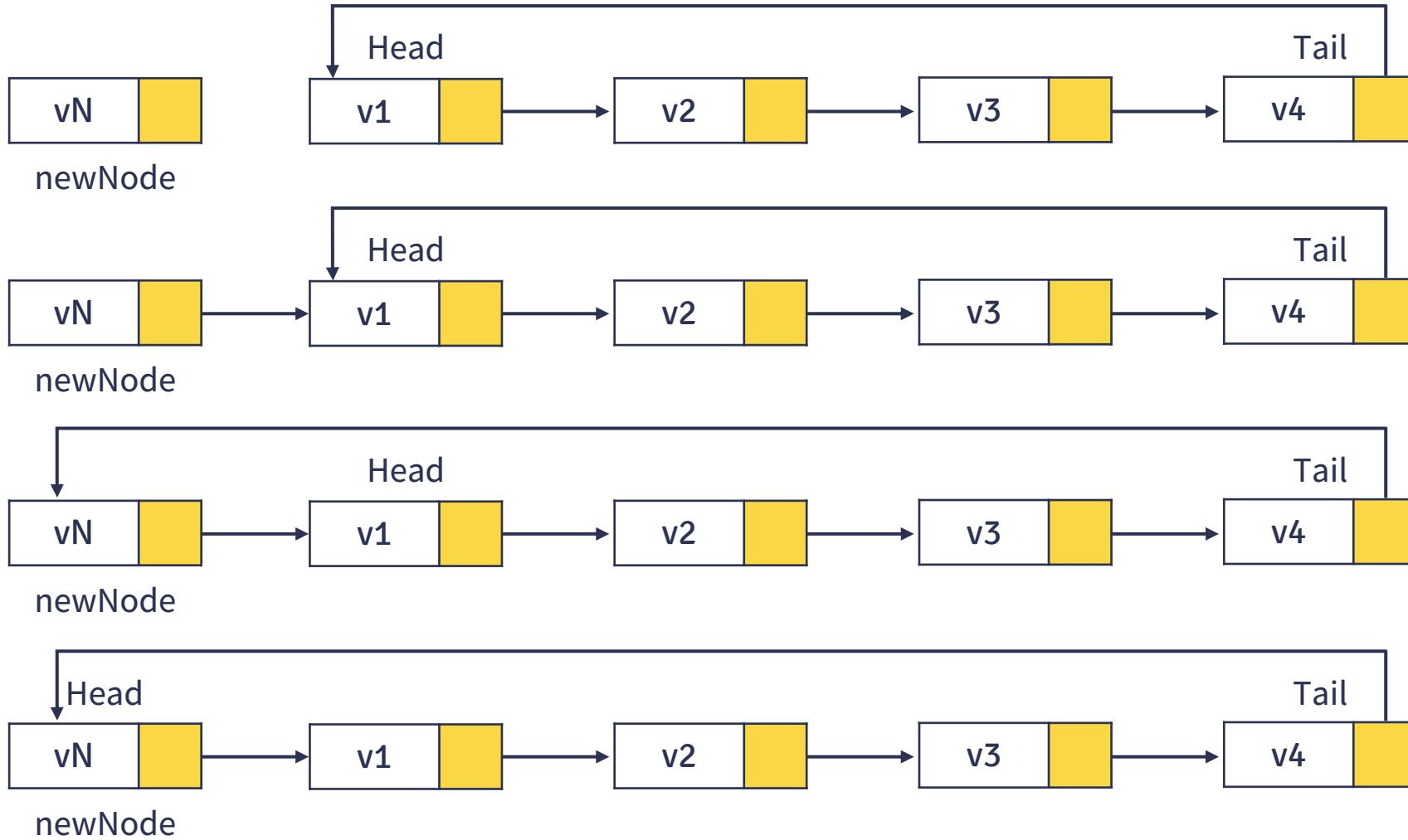
tail->dataName1 = val;
. .
tail->next = head;
*/
*/
```

Print Double Linked List ?

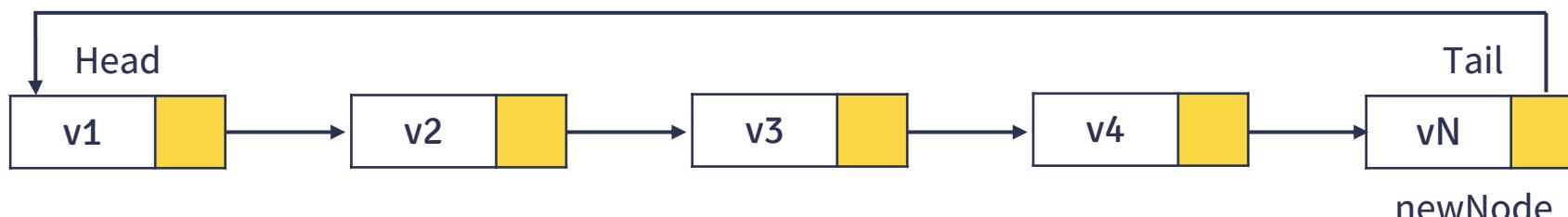
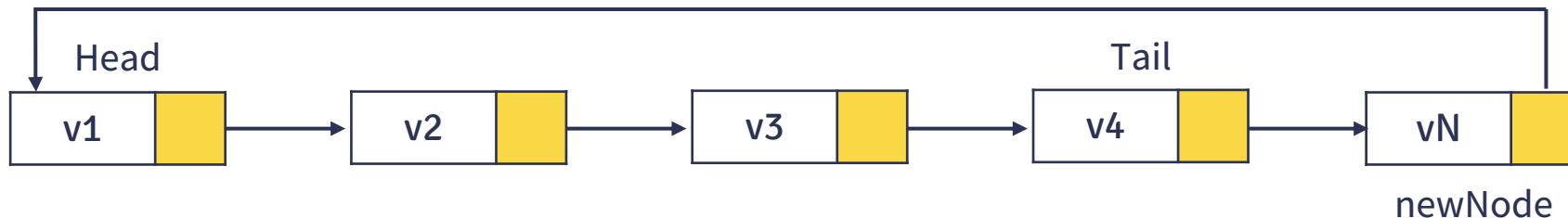
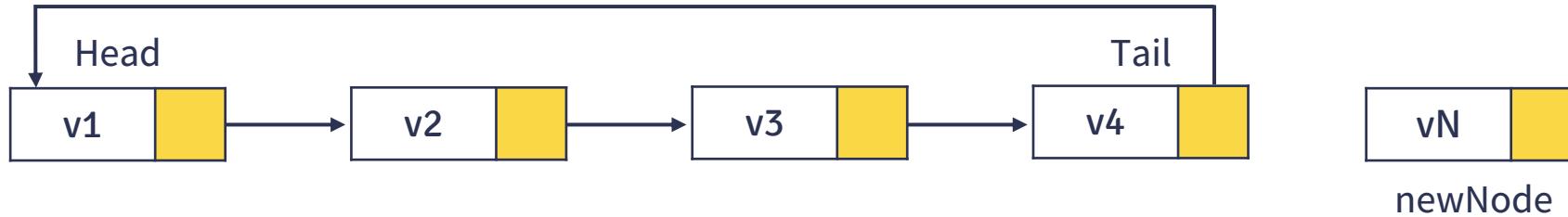
```
// Print Circular Single Linked List
/*
    LinkedListName *cur;
    cur = head;
    while( cur->next != head ){
        // Print Statement
        ....
        cur = cur->next;
    }
    // Print Last Node Statement
    . . .
*/
```



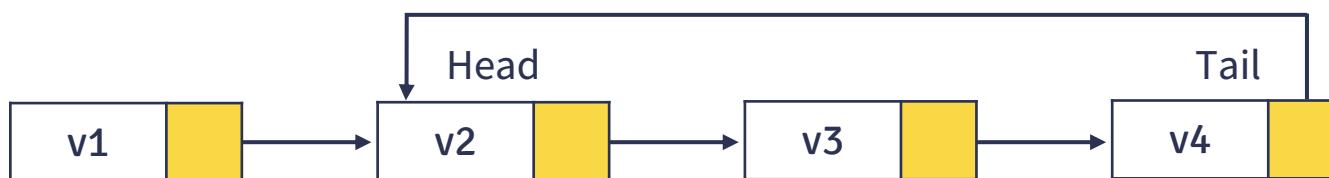
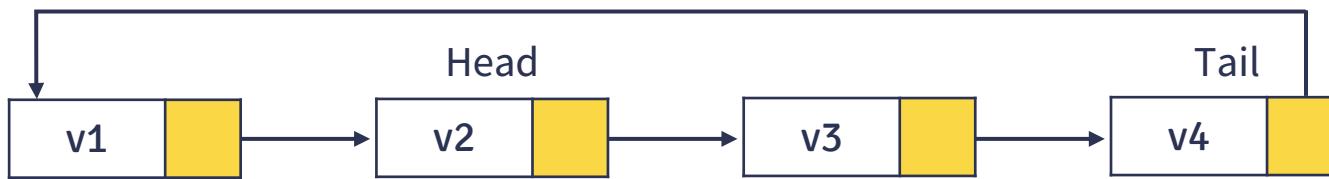
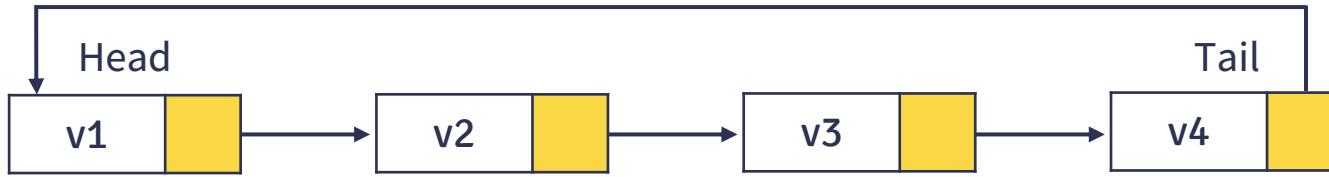
Added at Beginning Node ?



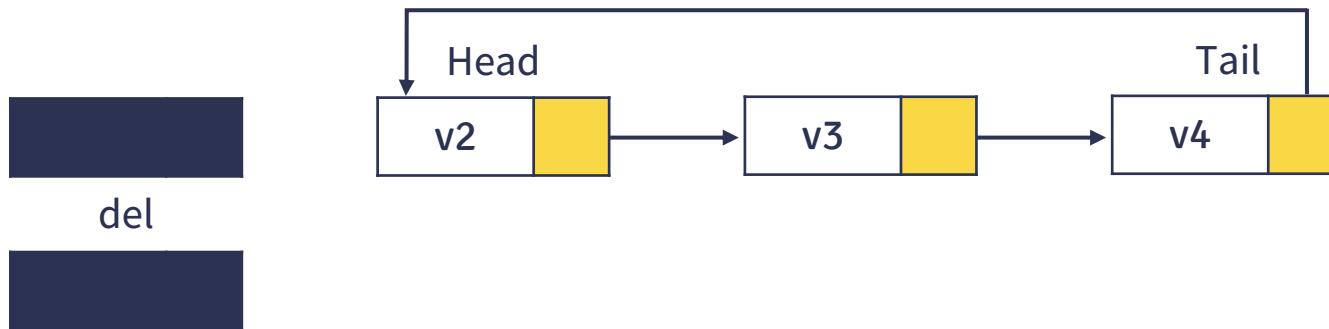
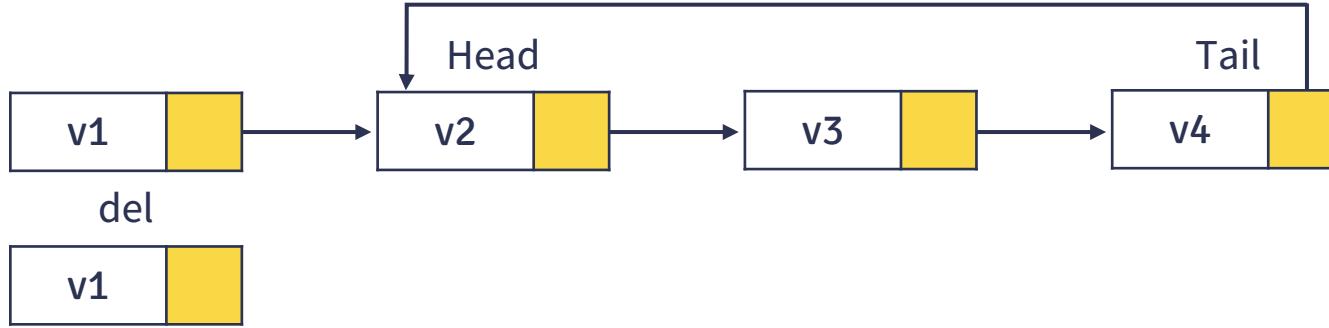
Added at Last Node ?



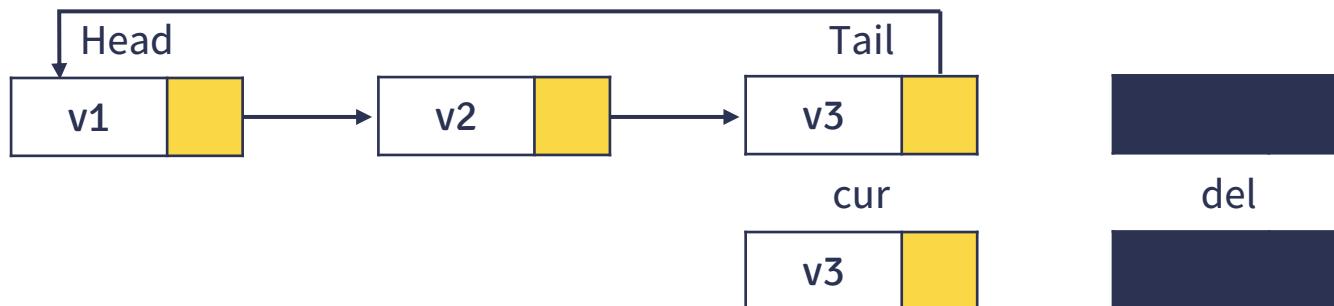
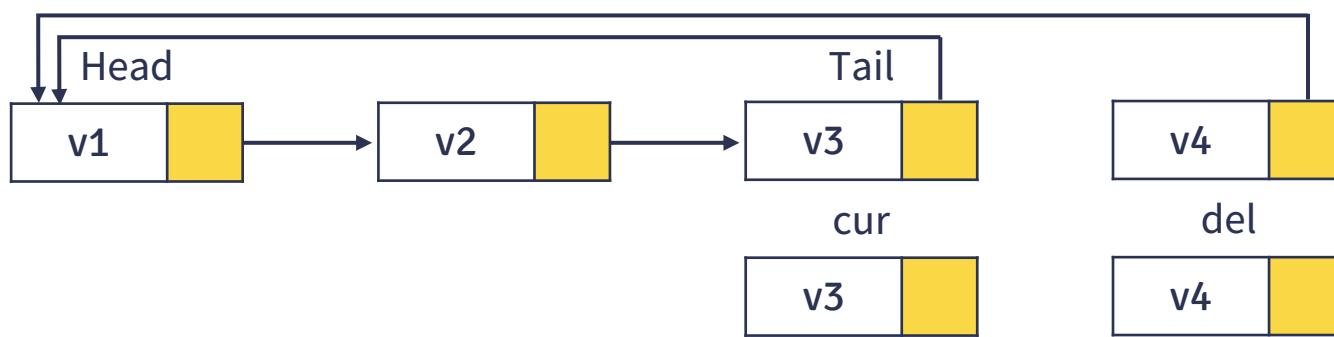
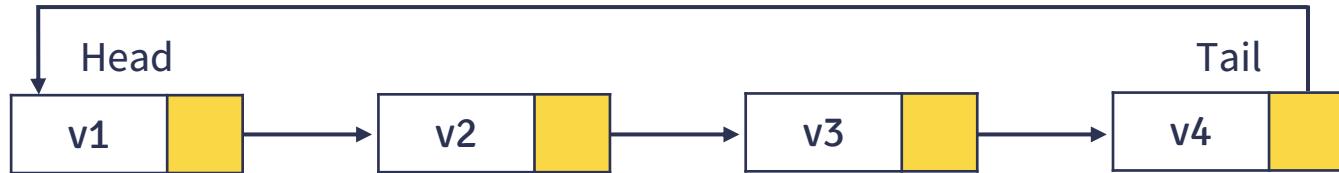
Delete the First Node ?



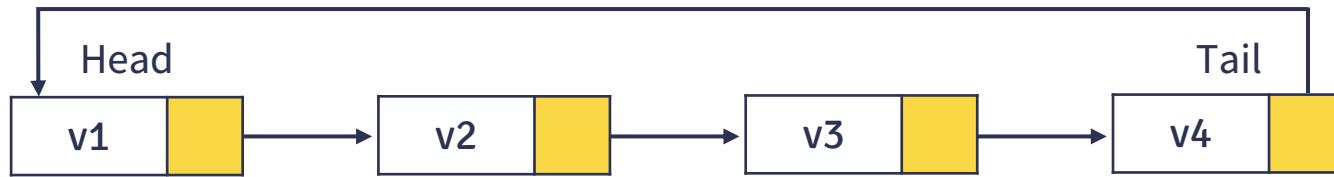
Delete the First Node ?



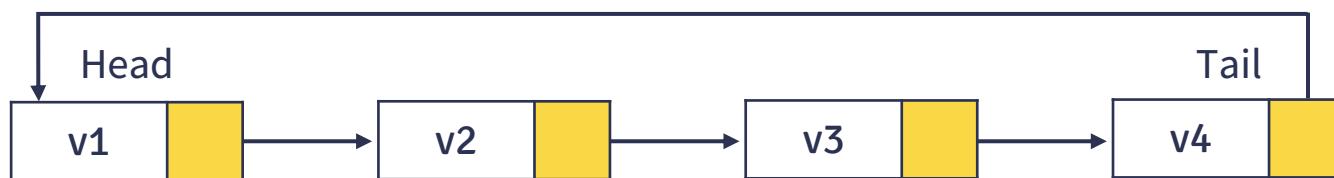
Delete the Last Node ?



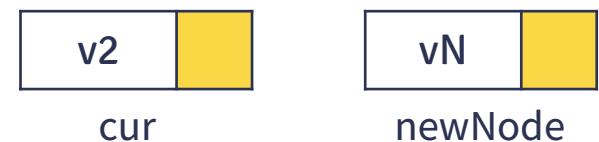
Added at Middle Node ?



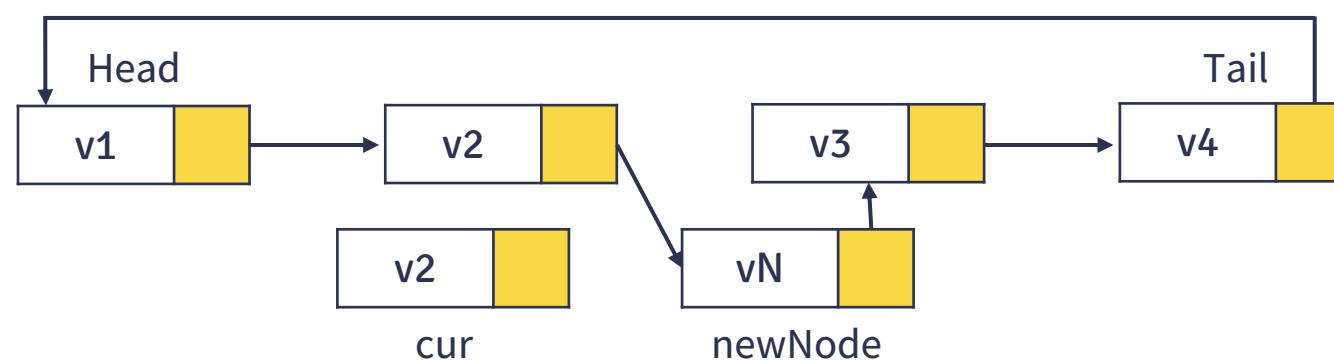
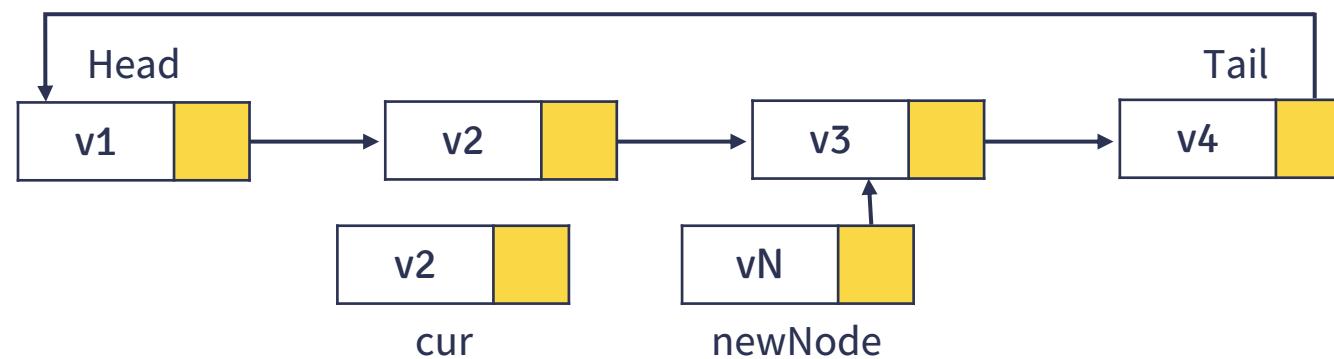
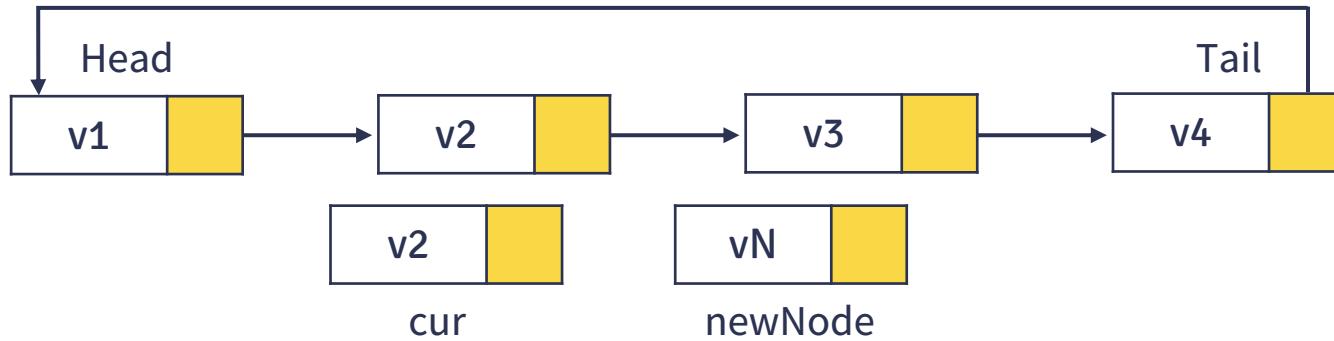
Tambah *vN* ke posisi 7



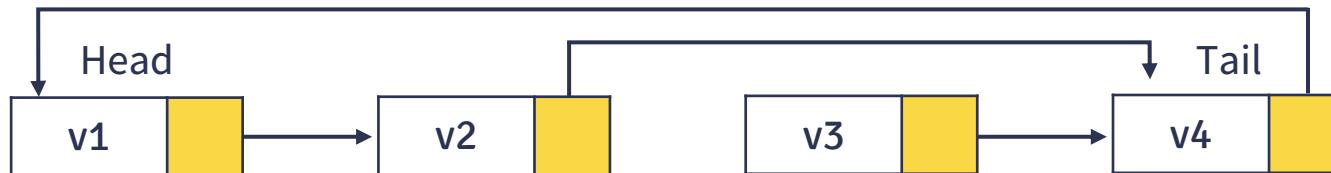
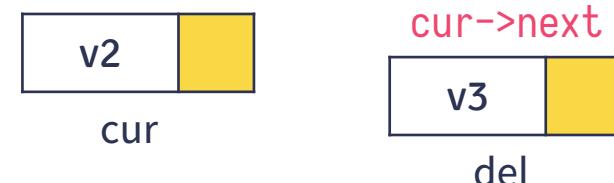
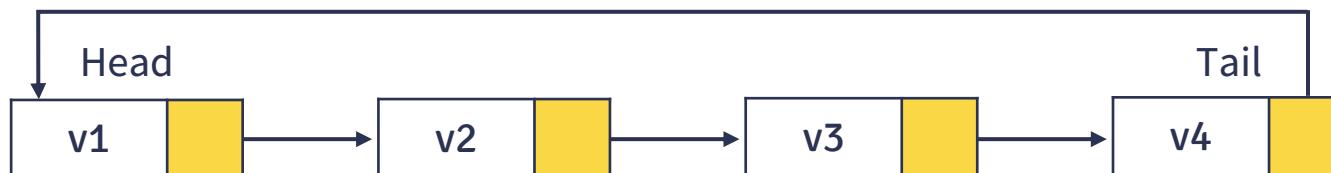
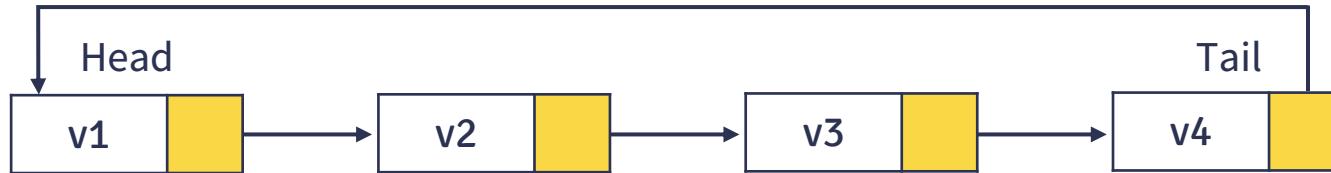
===== loop ($1 < \text{posisi} - 1$) =====>



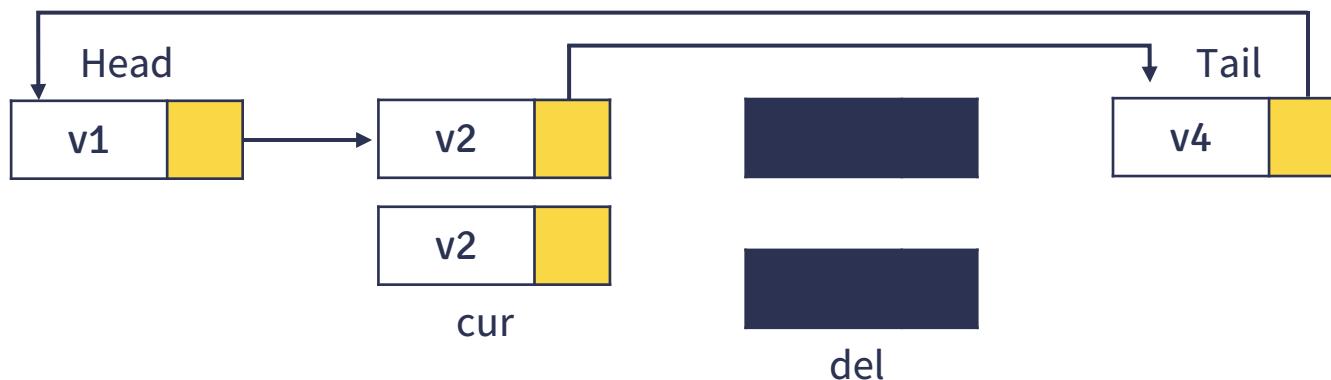
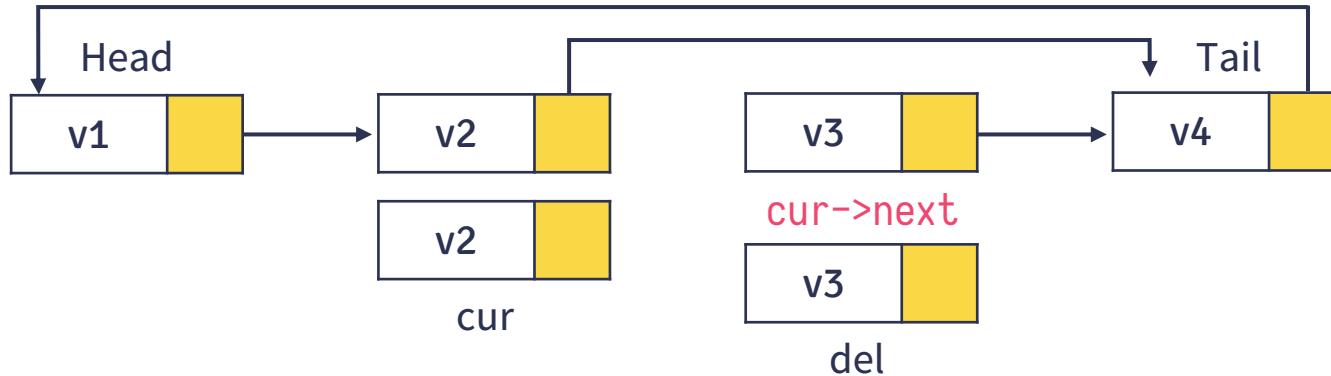
Added at Middle Node ?



Delete at Middle Node ?



Delete at Middle Node ?





Video Selanjutnya

Circular Double Linked List



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

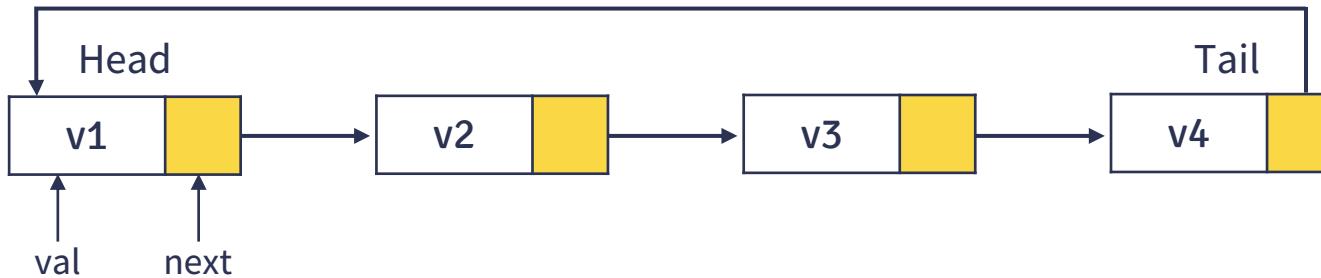
Saniati,S.ST.,M.T.

EPISODE **4C**

Circular Single Linked List

Circular Single Linked List ?

- Circular Linked List adalah sekumpulan node atau simpul yang tidak terdapat nilai NULL pada satupun nodenya.
- Bentuk node pada Circular Single Linked List sama dengan Single Linked List yang mempunyai data dan pointer next.
- Yang membedakan adalah jika pada Single Linked List node terakhir menunjuk ke NULL, tapi di Circular Single Linked List node terakhir menunjuk ke node pertama.



Deklarasi & Inisialisasi ?

```
// Deklarasi
/*
struct LinkedListName{
    typeData dataName1;
    . .
    LinkedListName *next;
};*/
*/
```

```
// Inisialisasi
/*
LinkedListName *head, *tail;
head = new LinkedListName();
tail = new LinkedListName();

head->dataName1 = val;
. .
head->next = tail;

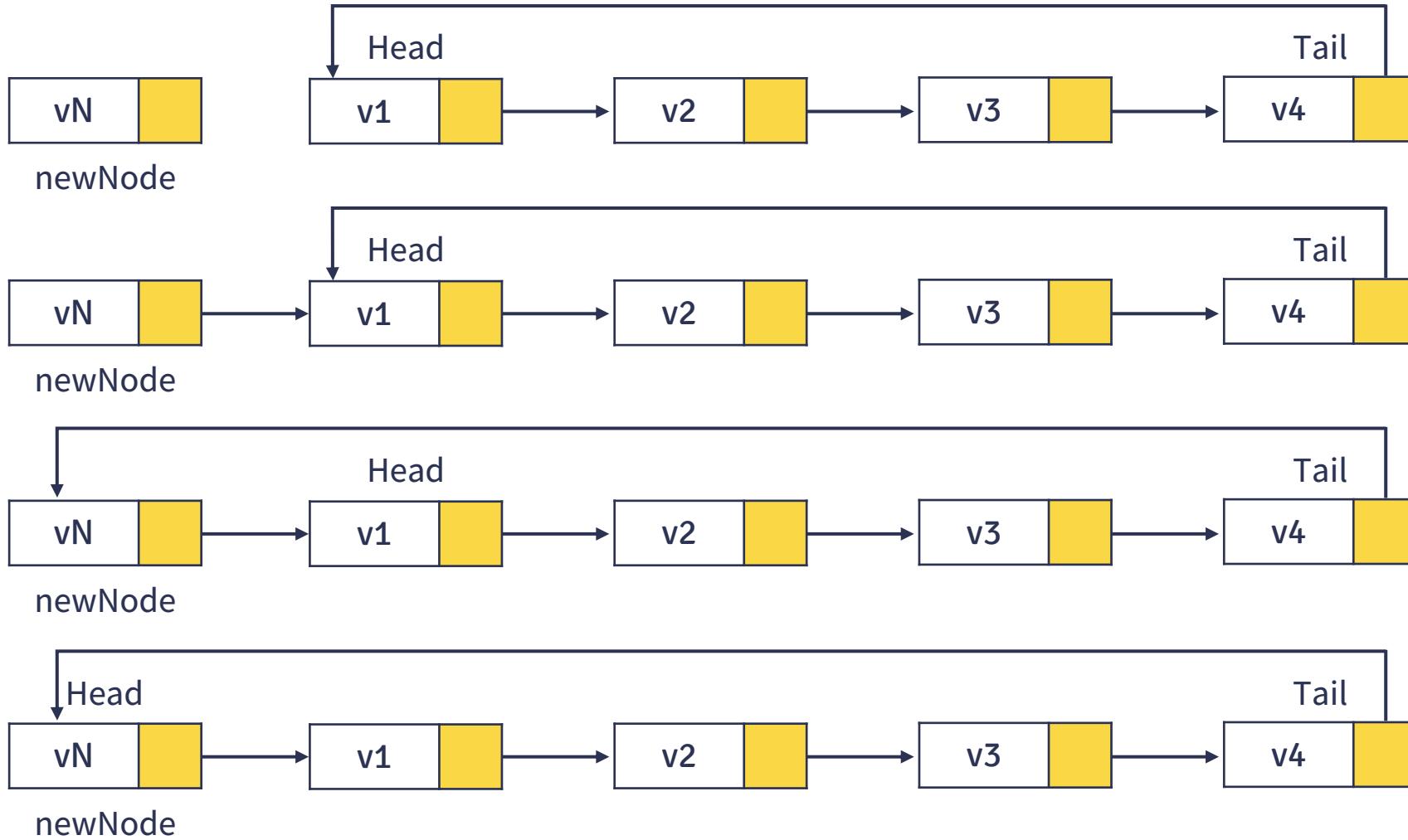
tail->dataName1 = val;
. .
tail->next = head;
*/
*/
```

Print Double Linked List ?

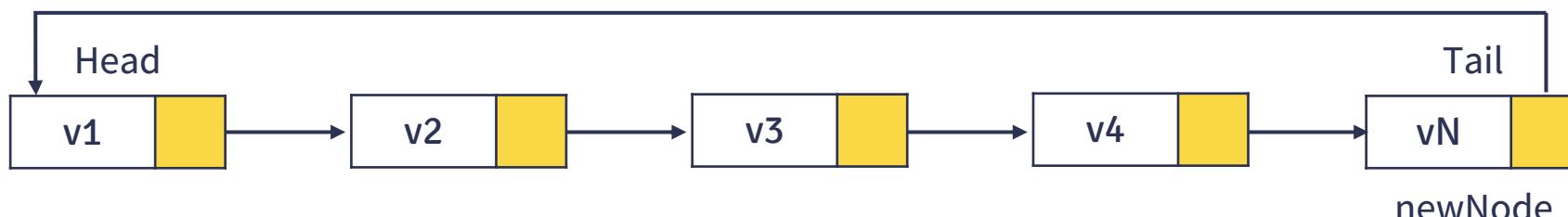
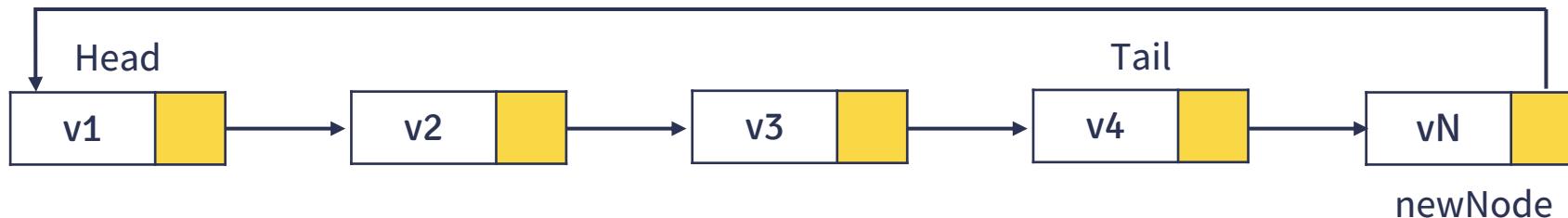
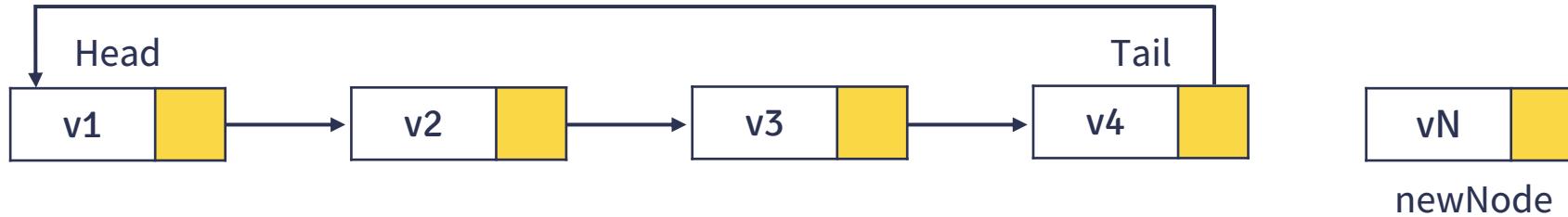
```
// Print Circular Single Linked List
/*
    LinkedListName *cur;
    cur = head;
    while( cur->next != head ){
        // Print Statement
        ....
        cur = cur->next;
    }
    // Print Last Node Statement
    . . .
*/
```



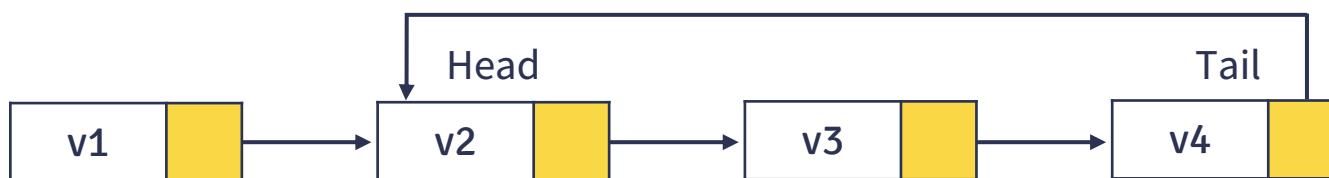
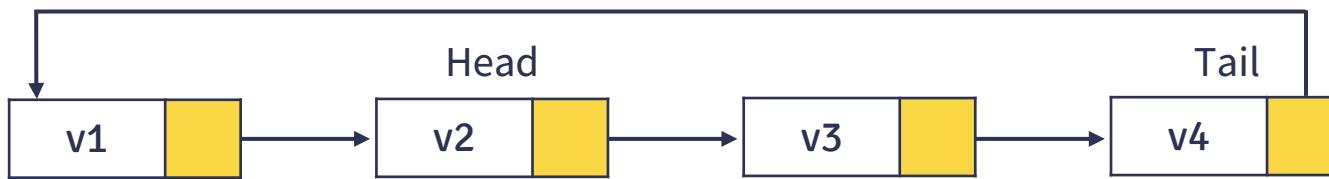
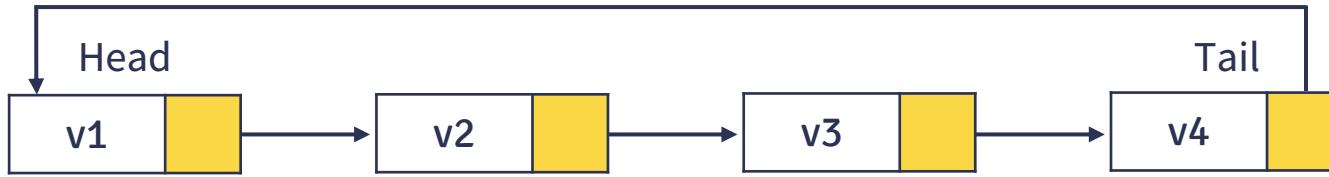
Added at Beginning Node ?



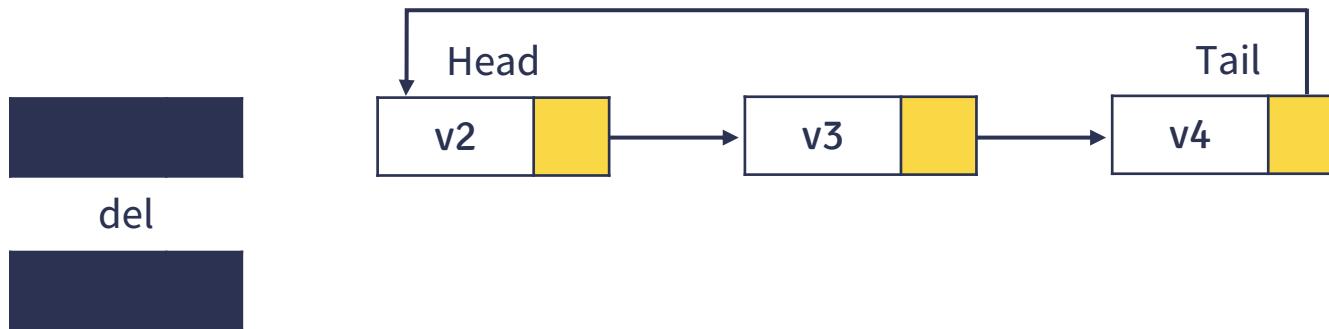
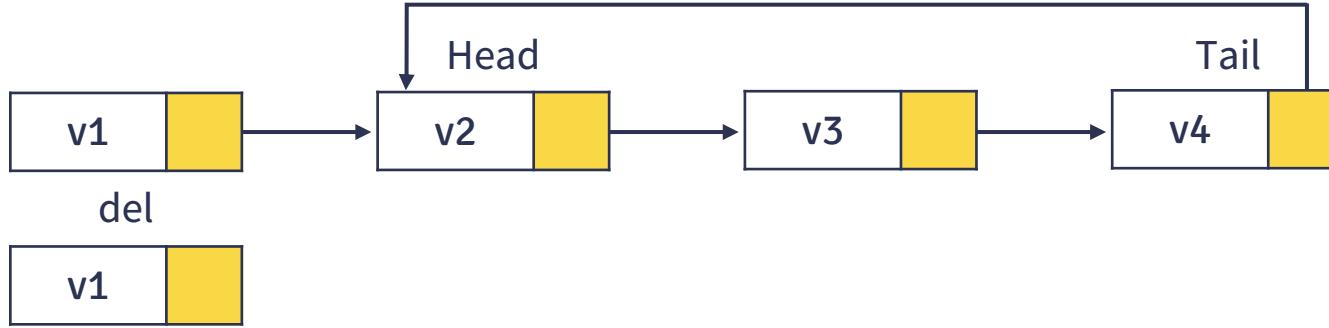
Added at Last Node ?



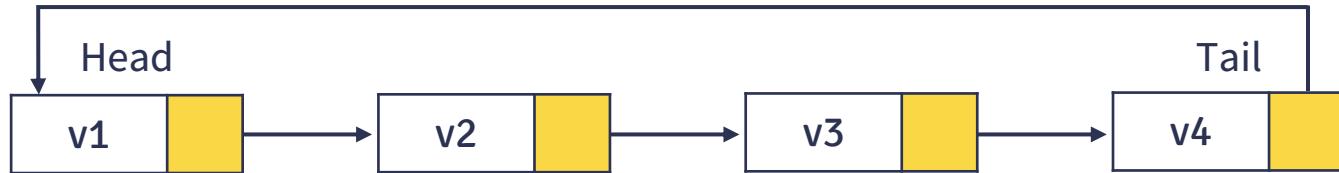
Delete the First Node ?



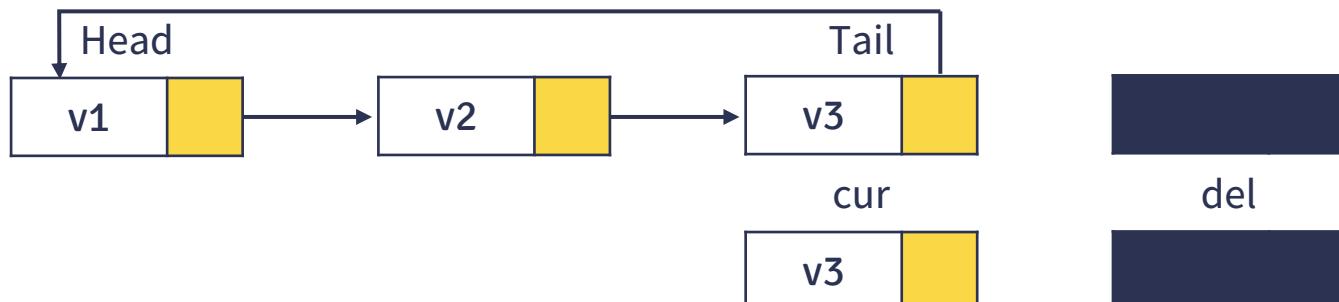
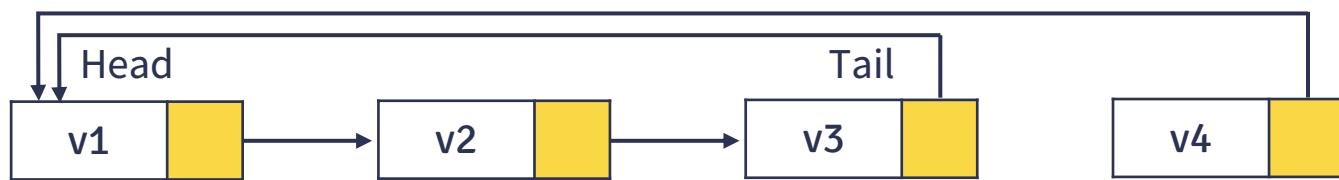
Delete the First Node ?



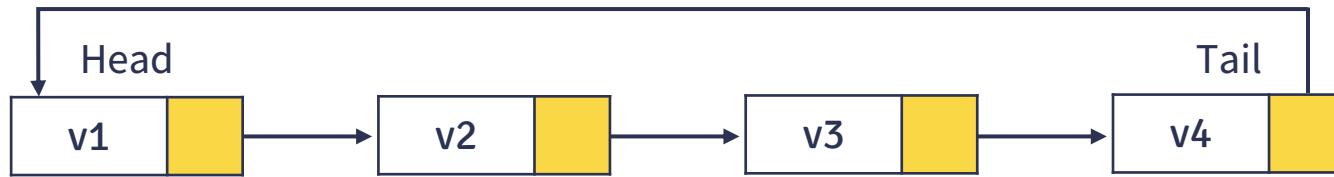
Delete the Last Node ?



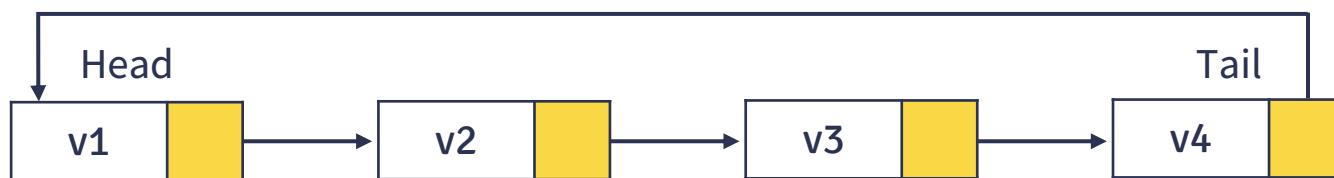
cur ======(cur->next != tail)=====>



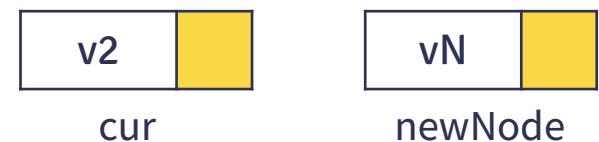
Added at Middle Node ?



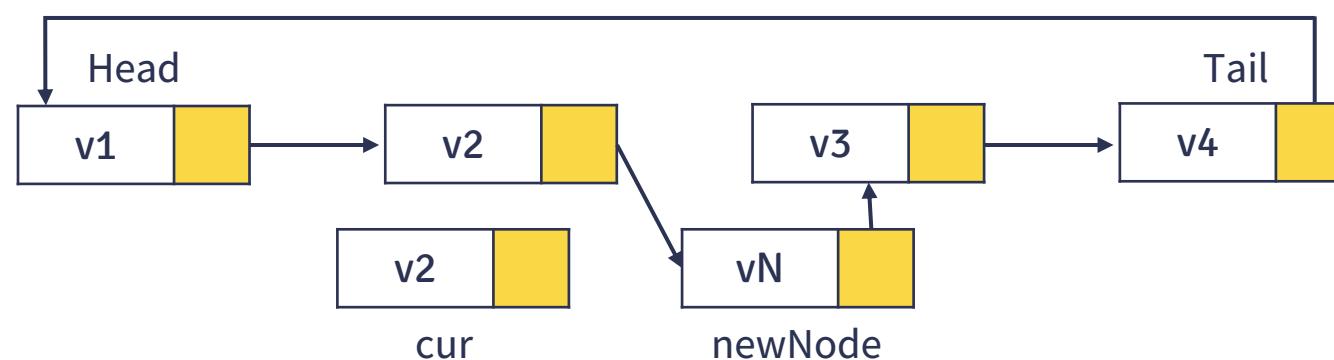
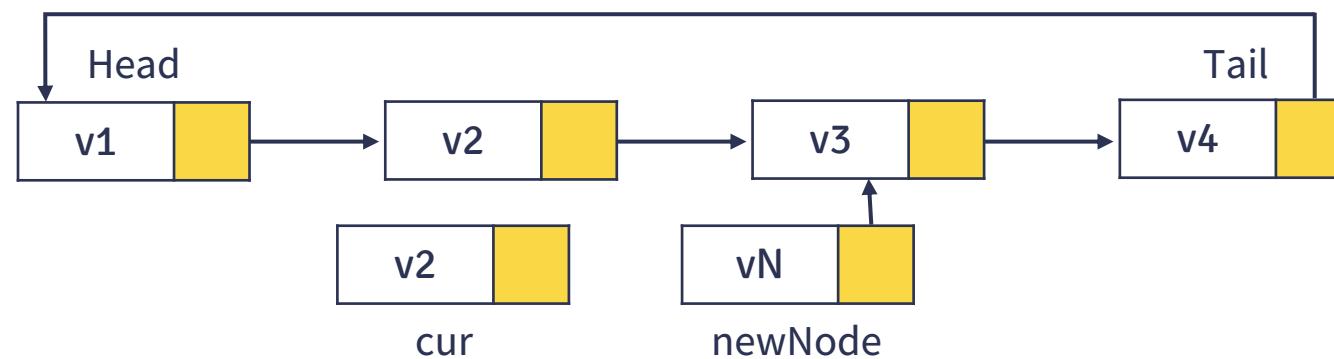
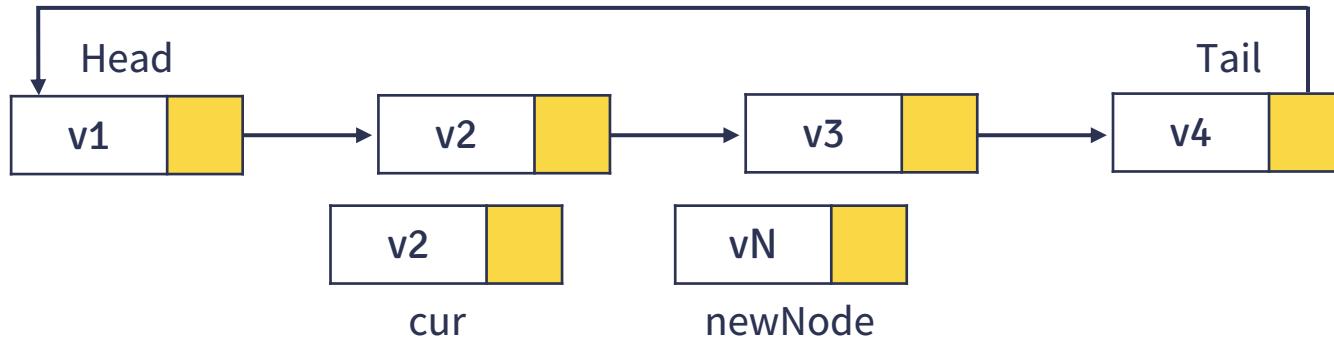
Tambah *vN* ke posisi 7



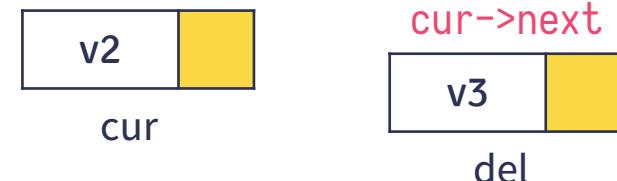
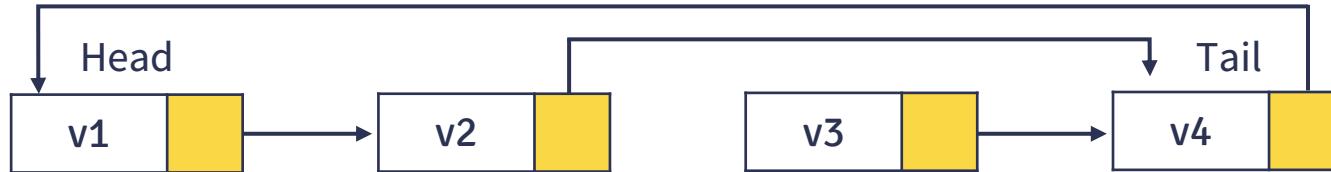
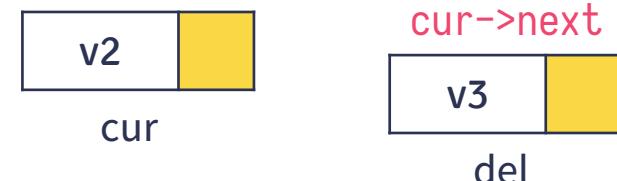
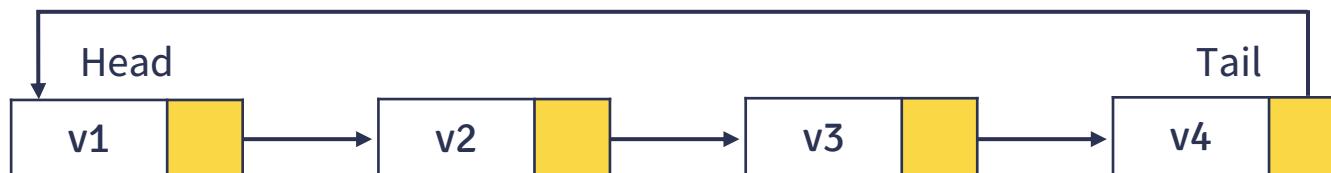
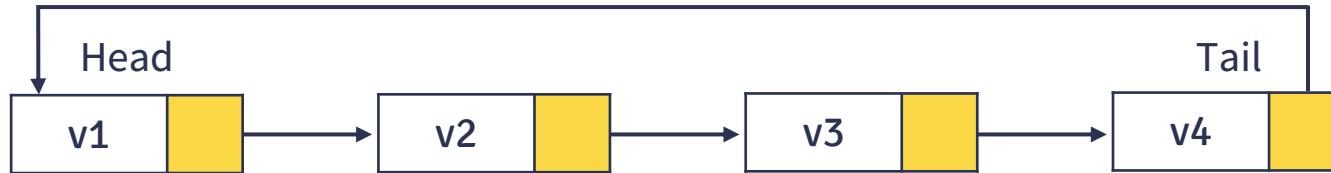
===== loop ($1 < \text{posisi} - 1$) =====>



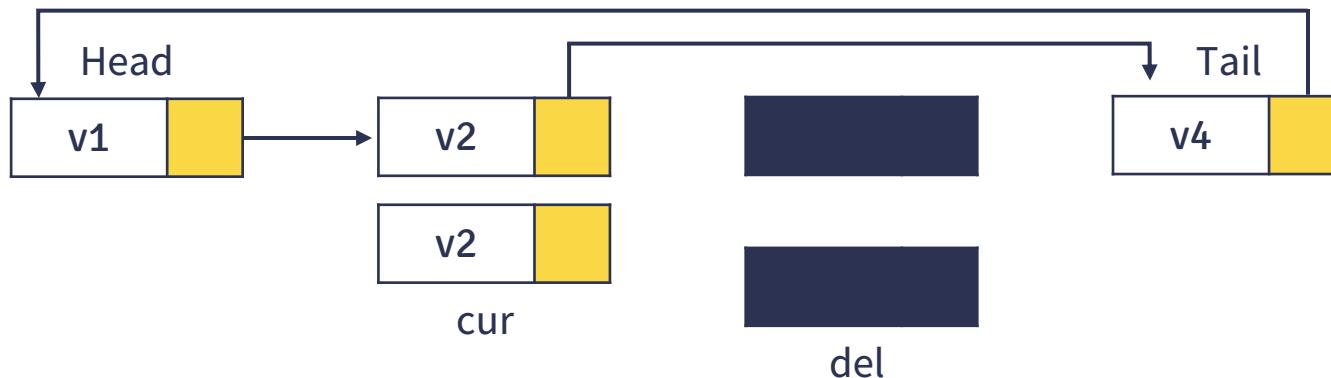
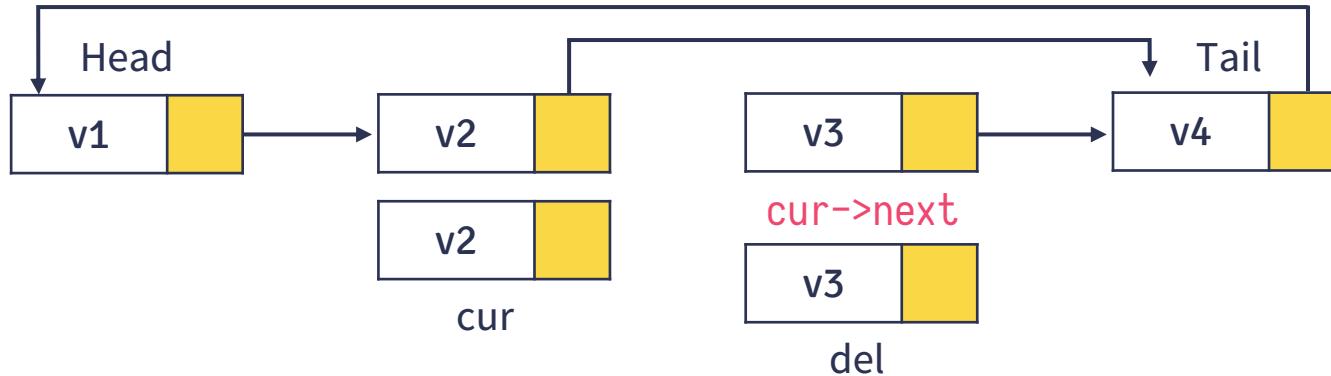
Added at Middle Node ?



Delete at Middle Node ?



Delete at Middle Node ?





Video Selanjutnya

Circular Double Linked List



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

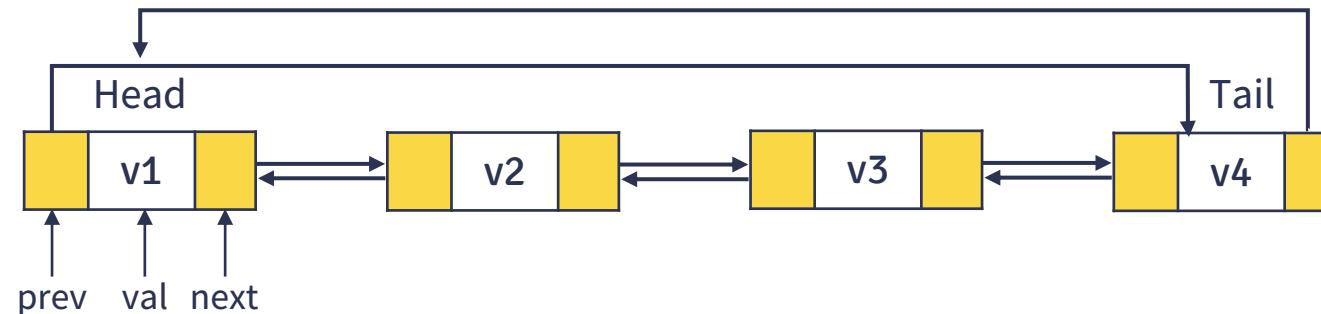
Saniati,S.ST.,M.T.

EPISODE **4D**

Circular Double Linked List

Circular Double Linked List ?

- Struktur node sama seperti Double Linked List yang mana terdapat data, pointer prev dan pointer next.
- Karena Circular linked list maka pointer next pada tail menunjuk ke head dan pointer prev pada head menunjuk ke tail.



Deklarasi & Inisialisasi ?

```
/*
struct LinkListName{
    // komponen / member
    dataTypeData1 dataName1;
    . . .
    LinkListName *prev;
    LinkListName *next;
};
*/
```

```
/*
LinkListName *head, *tail;
head = (LinkListName*) malloc(sizeof(LinkListName));
tail = new LinkListName();

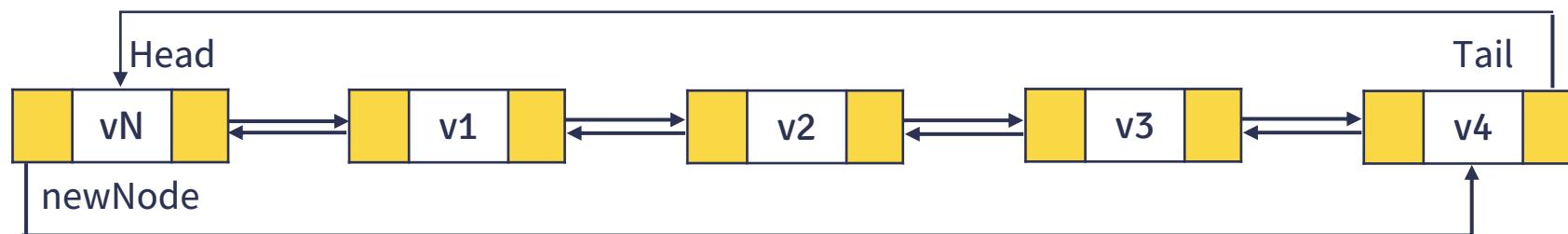
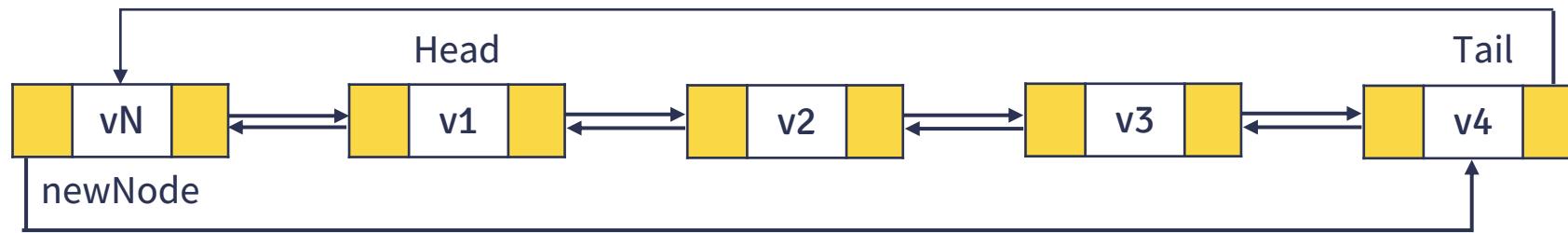
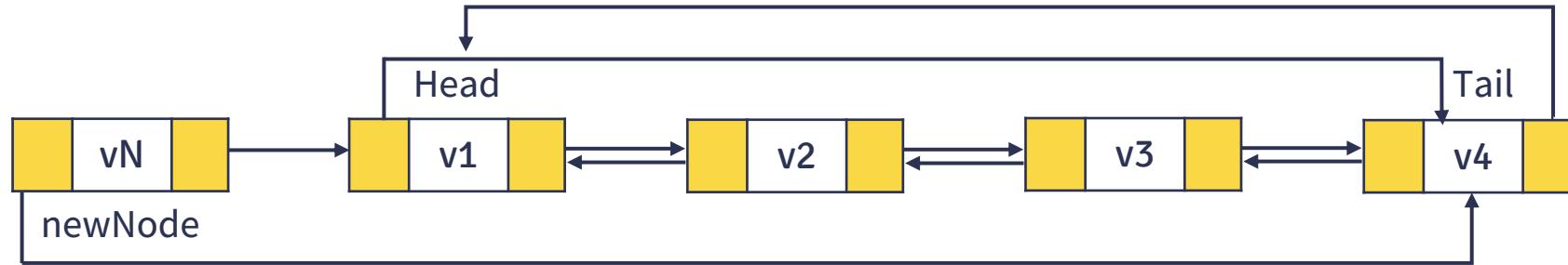
*/
/*
head->dataName1 = valData1;
. . .
head->prev = tail;
head->next = tail;

tail->dataName1 = valData1;
. . .
tail->prev = head;
tail->next = head;
*/
*/
```

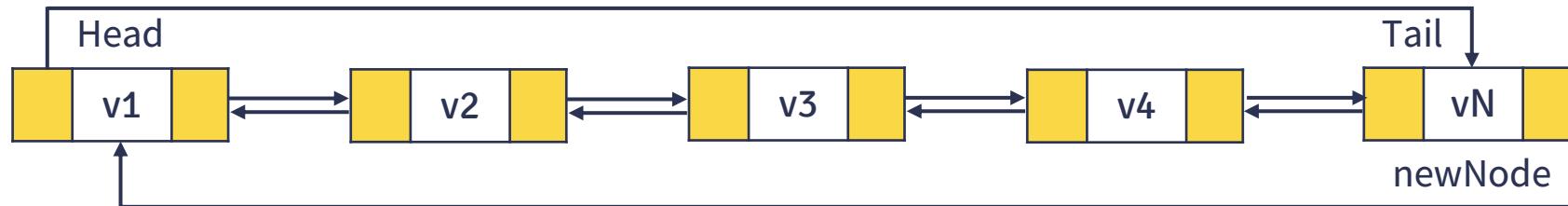
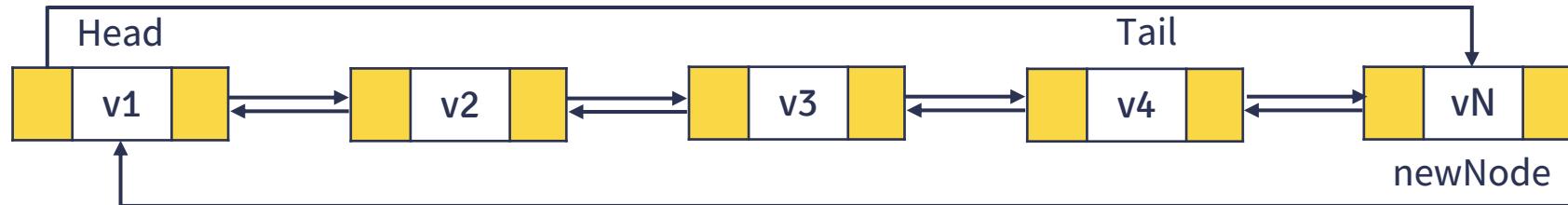
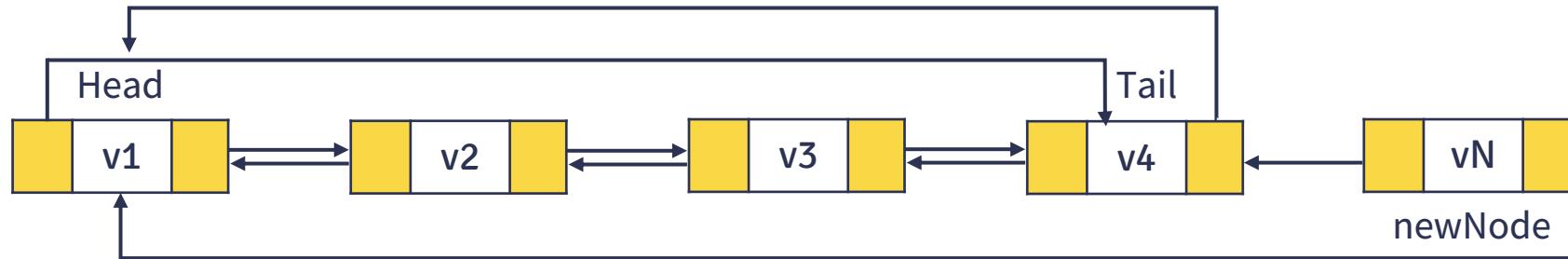
Print Circular Double Linked List ?

```
/*
LinkListName *cur;
cur = head;
while( cur->next != head ){
    // print
    cur = cur->next;
}
// print last node
*/
```

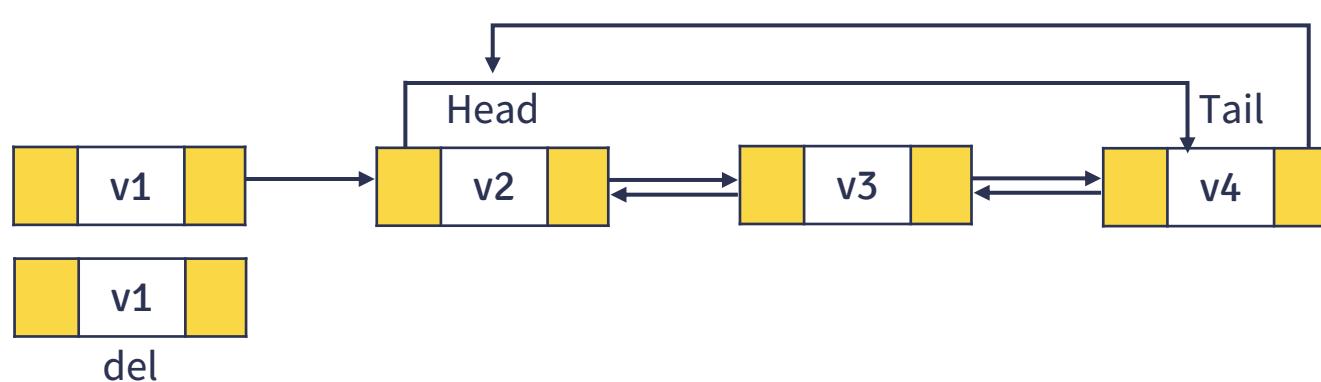
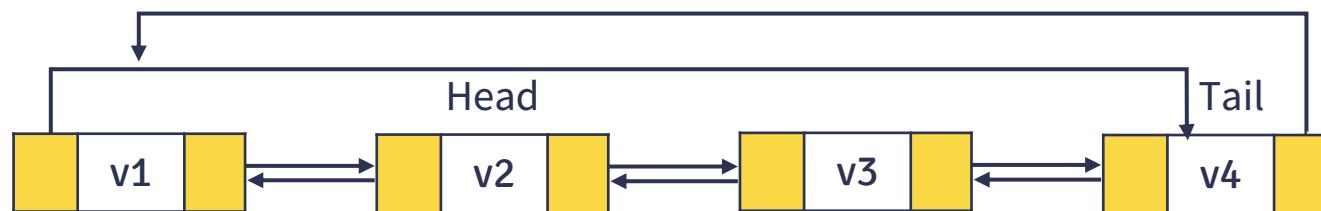
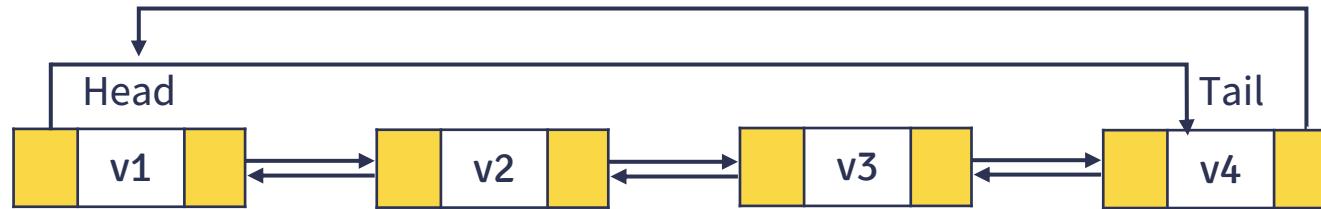
Added at Beginning Node ?



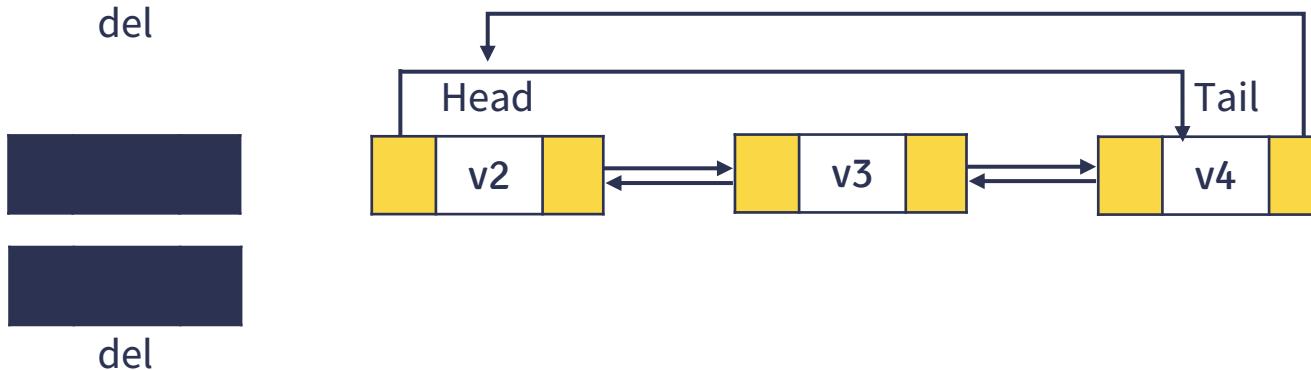
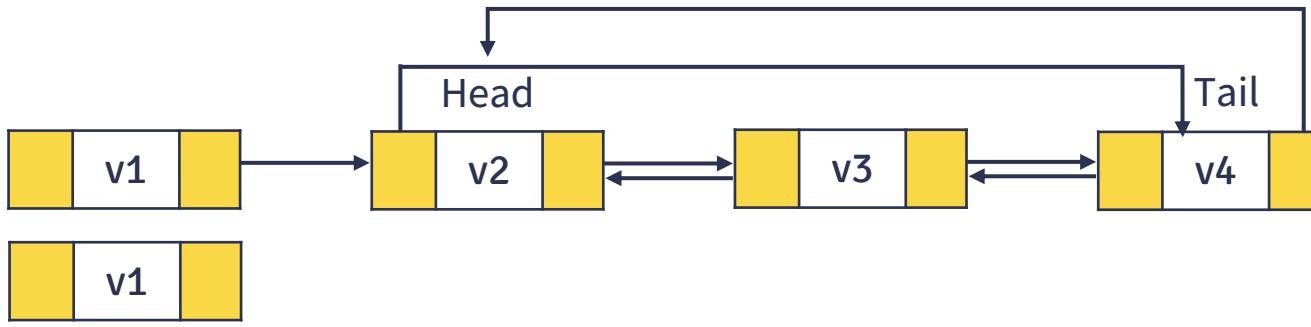
Added at Last Node ?



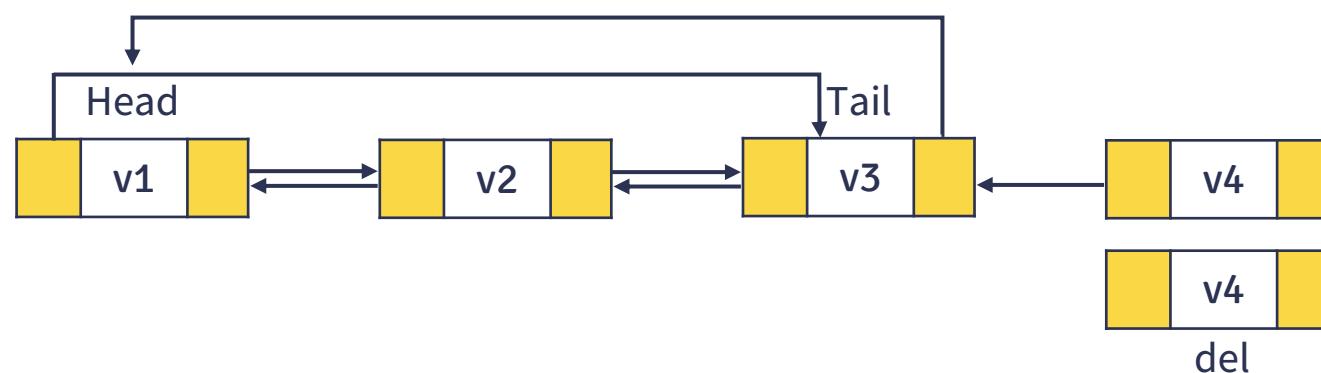
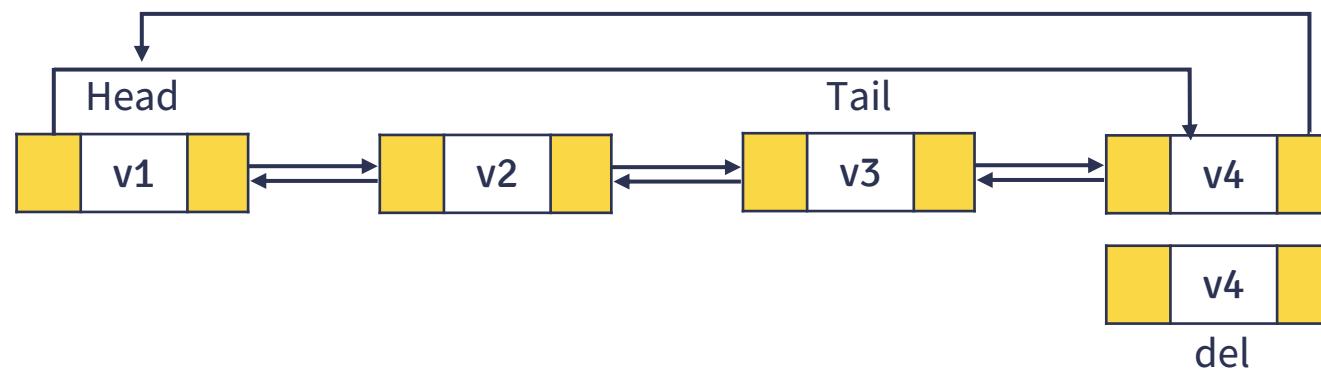
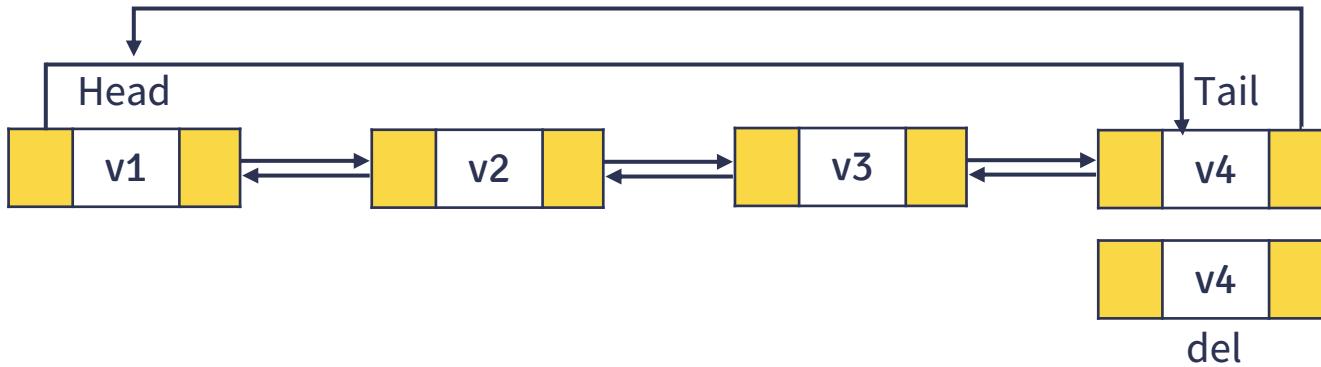
Delete the First Node ?



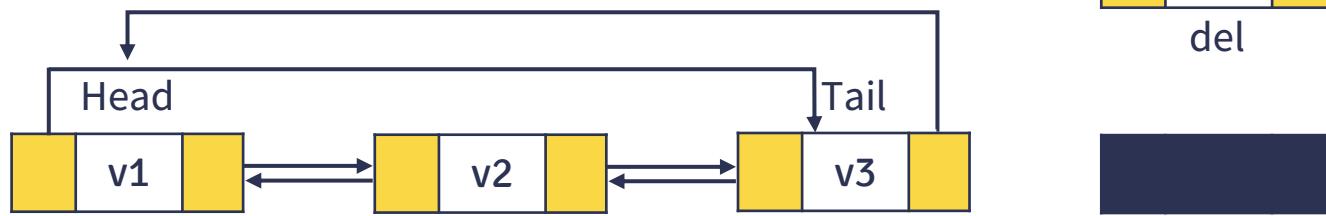
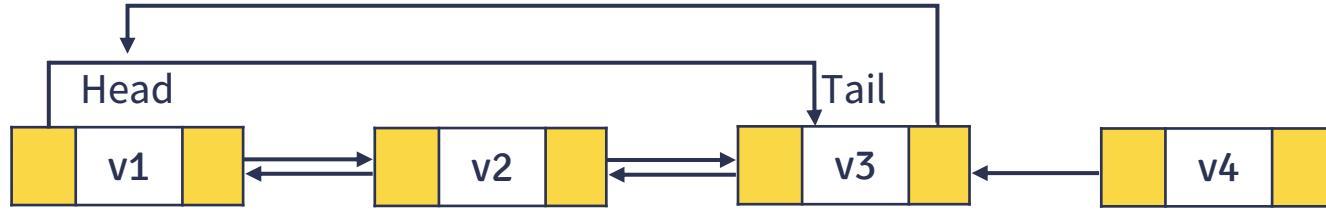
Delete the First Node ?



Delete the Last Node ?



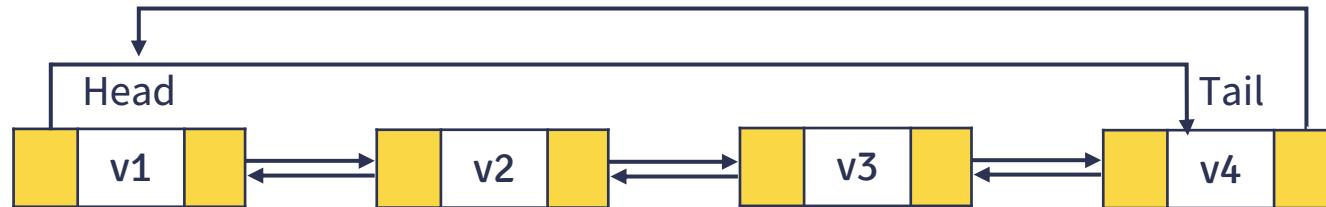
Delete the Last Node ?



v4
del

del

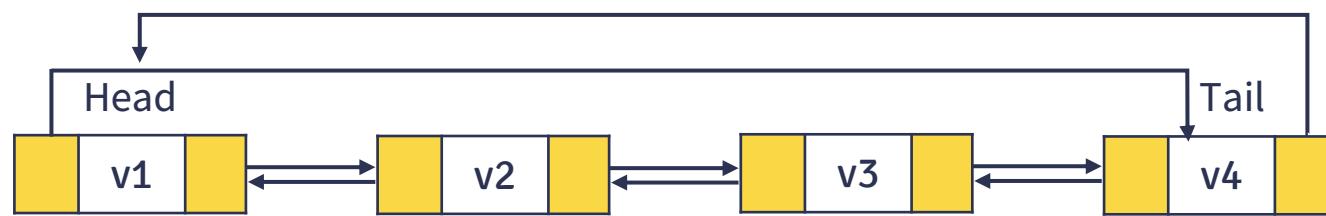
Added at Middle Node ?



Tambah ke posisi 7



newNode



===== loop ($1 < \text{posisi} - 1$) =====

$\text{cur} \rightarrow \text{next}$



cur

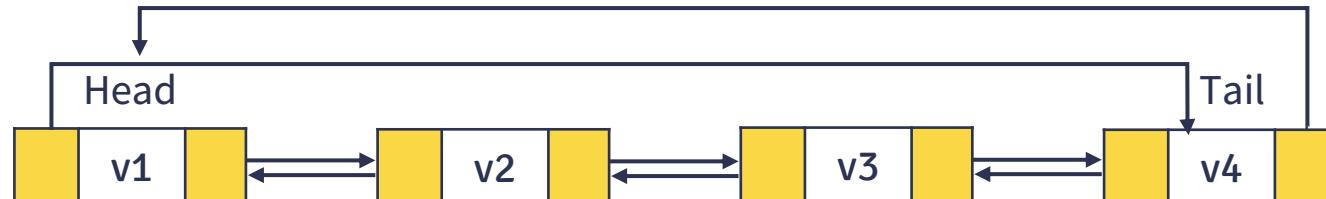


newNode



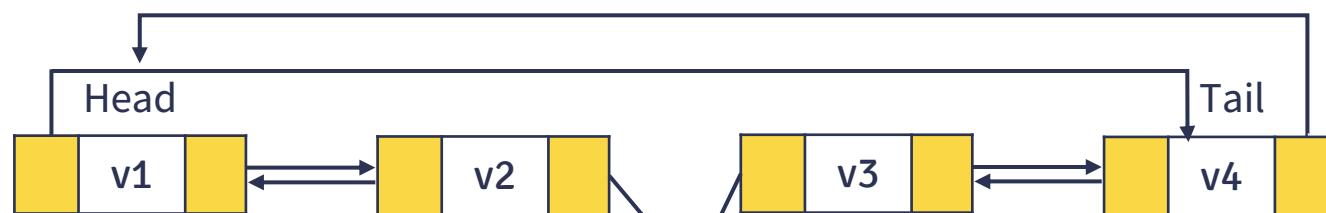
afterNode

Added at Middle Node ?

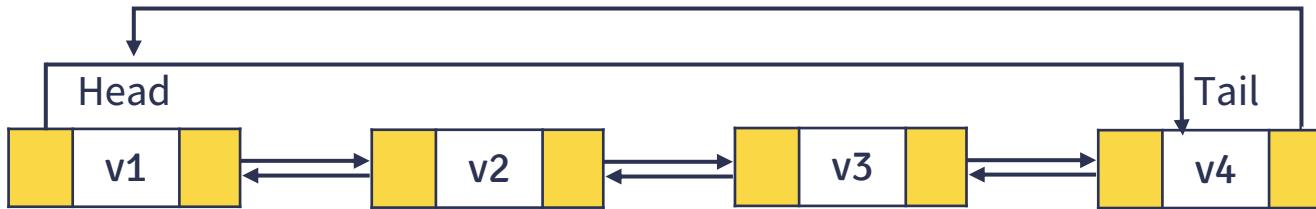


===== loop ($1 < \text{posisi} - 1$) =====>

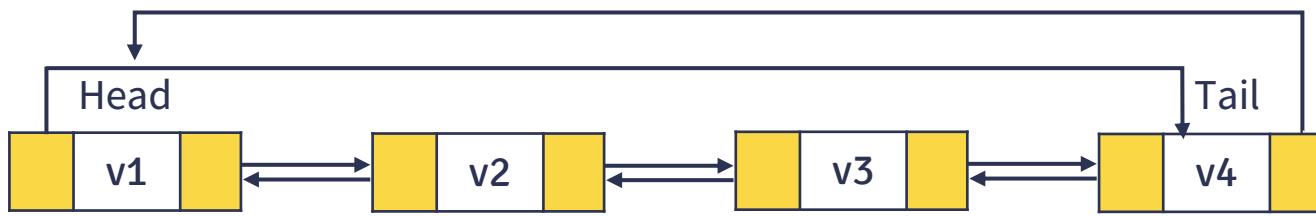
cur->next



Delete at Middle Node ?

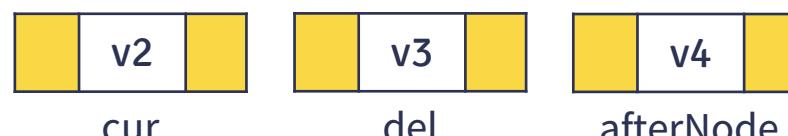


Hapus posisi ke-7



===== loop ($1 < \text{posisi} - 1$) =====>

cur->next del->next

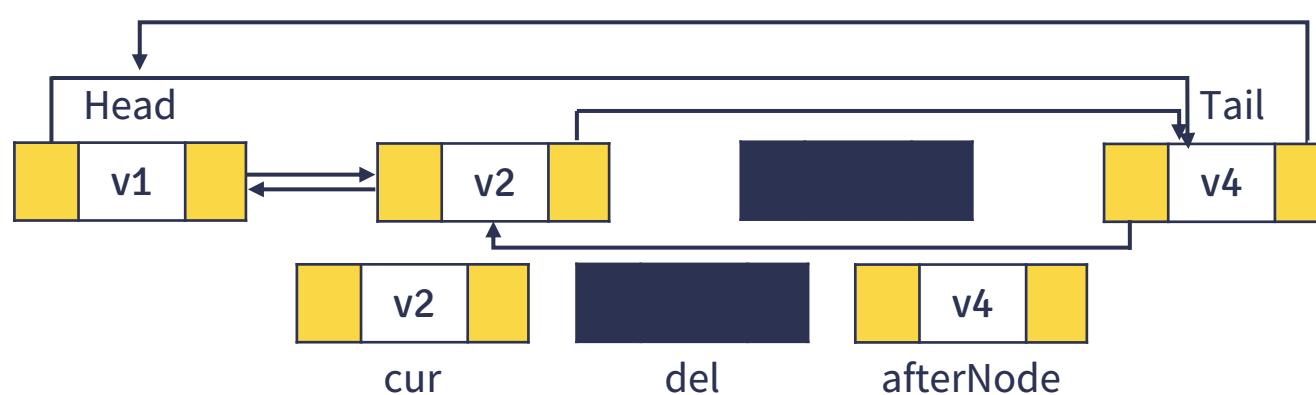
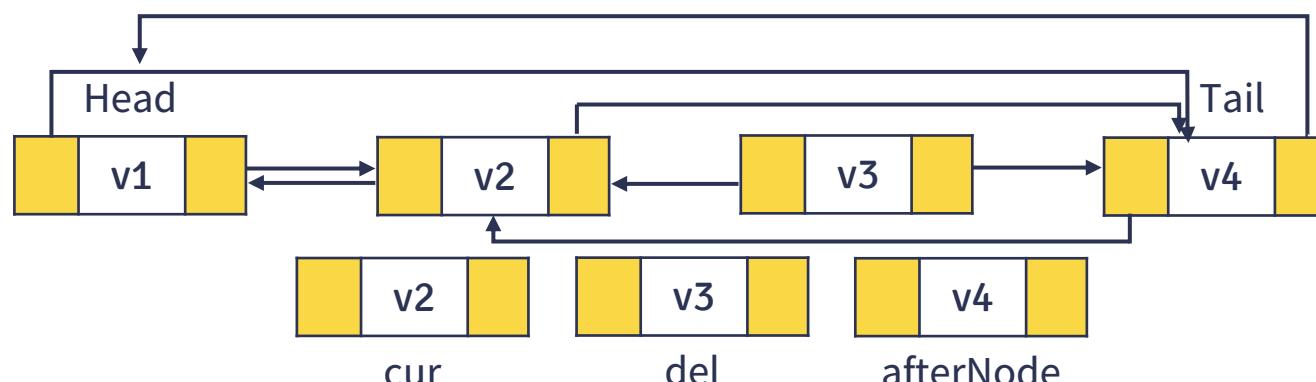
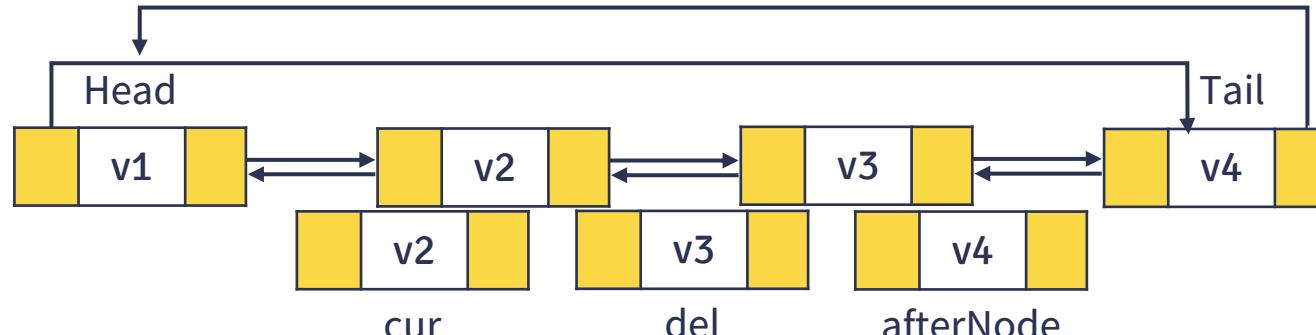


cur

del

afterNode

Delete at Middle Node ?





Video Selanjutnya

Stack (tumpukan)



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

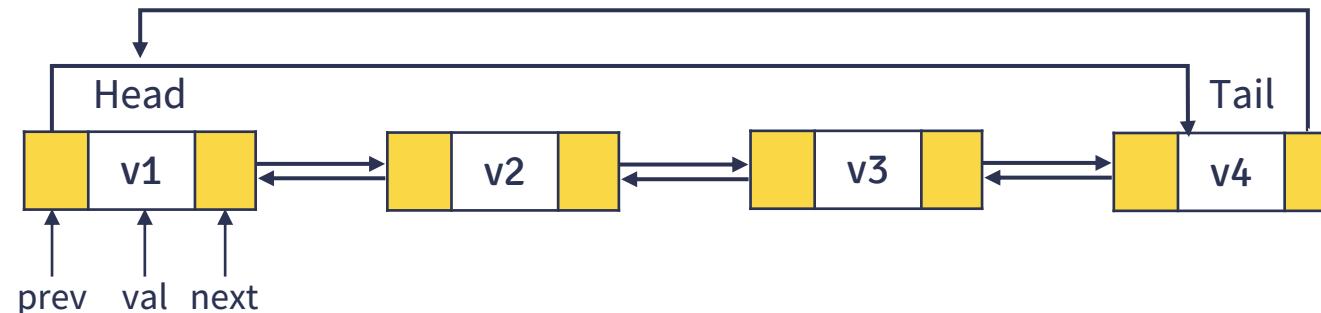
Saniati,S.ST.,M.T.

EPISODE **4D**

Circular Double Linked List

Circular Double Linked List ?

- Struktur node sama seperti Double Linked List yang mana terdapat data, pointer prev dan pointer next.
- Karena Circular linked list maka pointer next pada tail menunjuk ke head dan pointer prev pada head menunjuk ke tail.



Deklarasi & Inisialisasi ?

```
/*
struct LinkListName{
    // komponen / member
    dataTypeData1 dataName1;
    . . .
    LinkListName *prev;
    LinkListName *next;
};
*/
```

```
/*
LinkListName *head, *tail;
head = (LinkListName*) malloc(sizeof(LinkListName));
tail = new LinkListName();

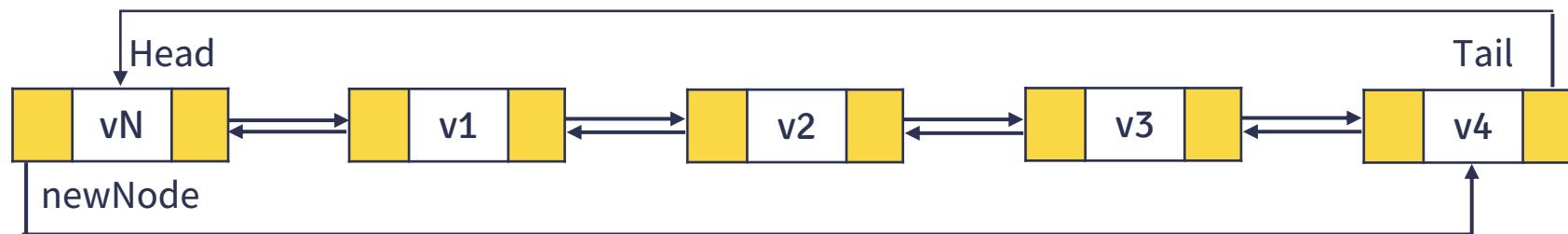
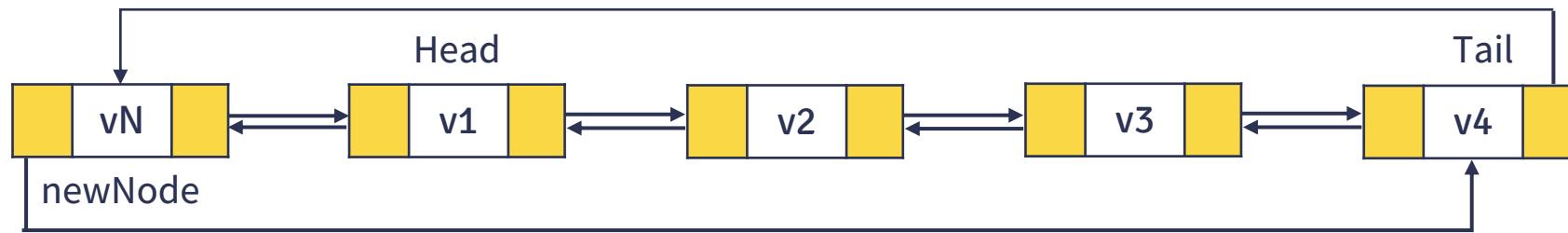
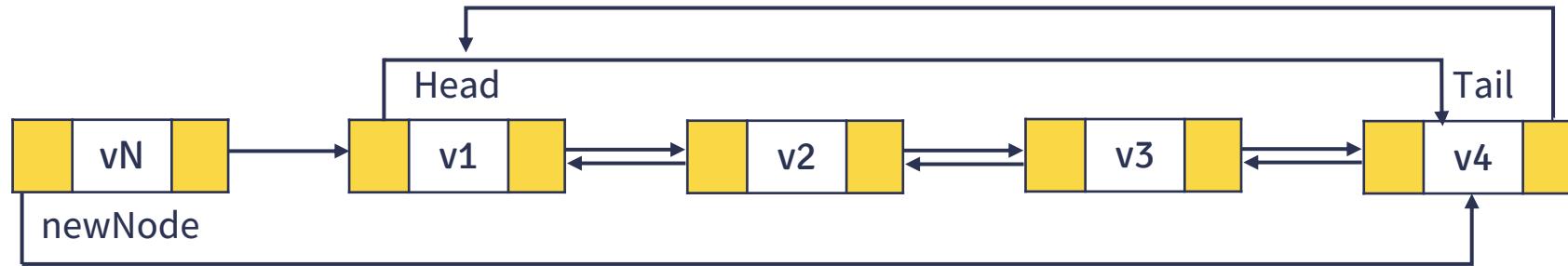
*/
/*
head->dataName1 = valData1;
. . .
head->prev = tail;
head->next = tail;

tail->dataName1 = valData1;
. . .
tail->prev = head;
tail->next = head;
*/
*/
```

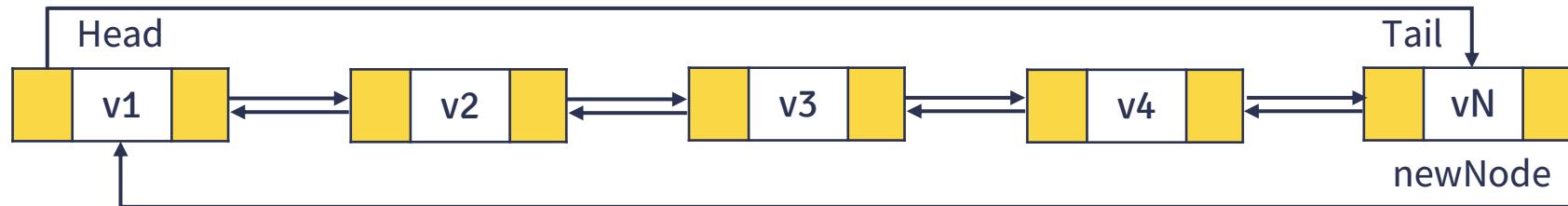
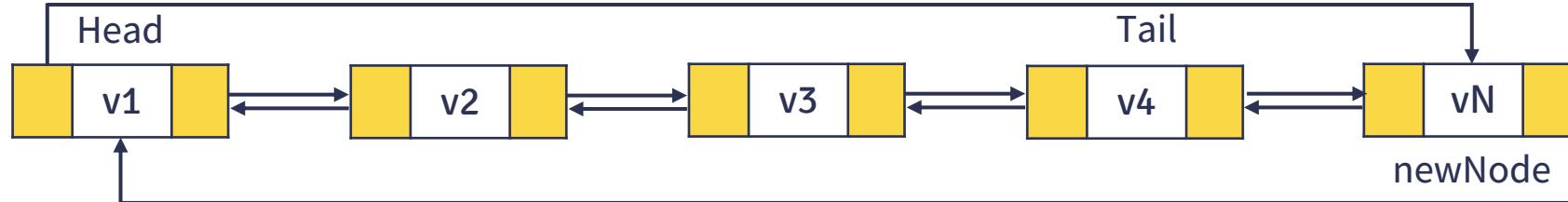
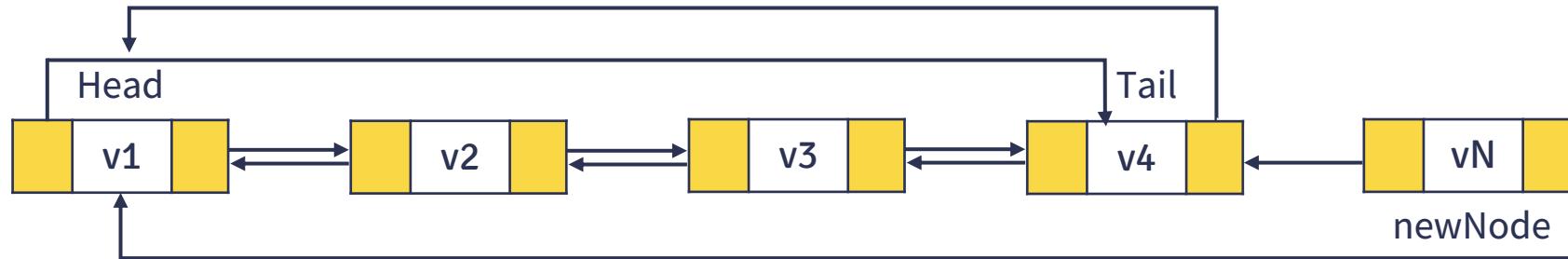
Print Circular Double Linked List ?

```
/*
LinkListName *cur;
cur = head;
while( cur->next != head ){
    // print
    cur = cur->next;
}
// print last node
*/
```

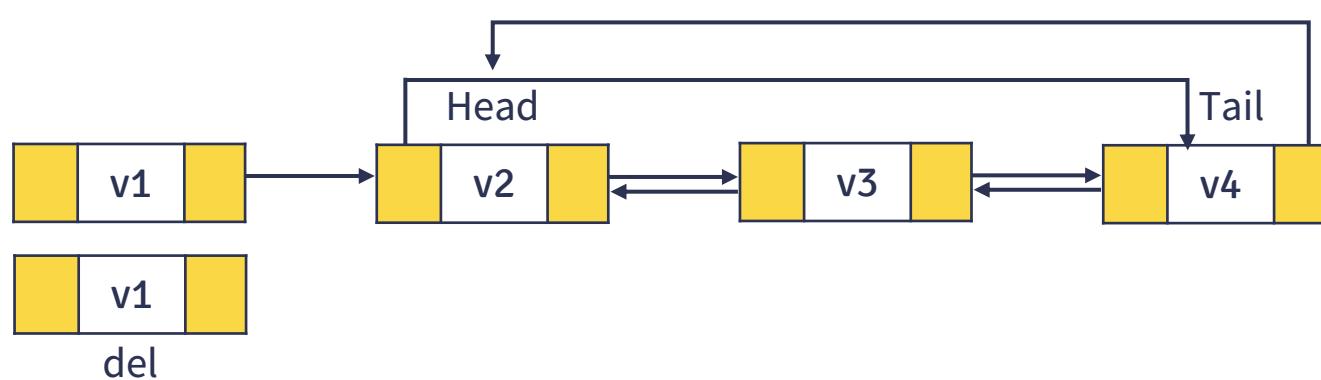
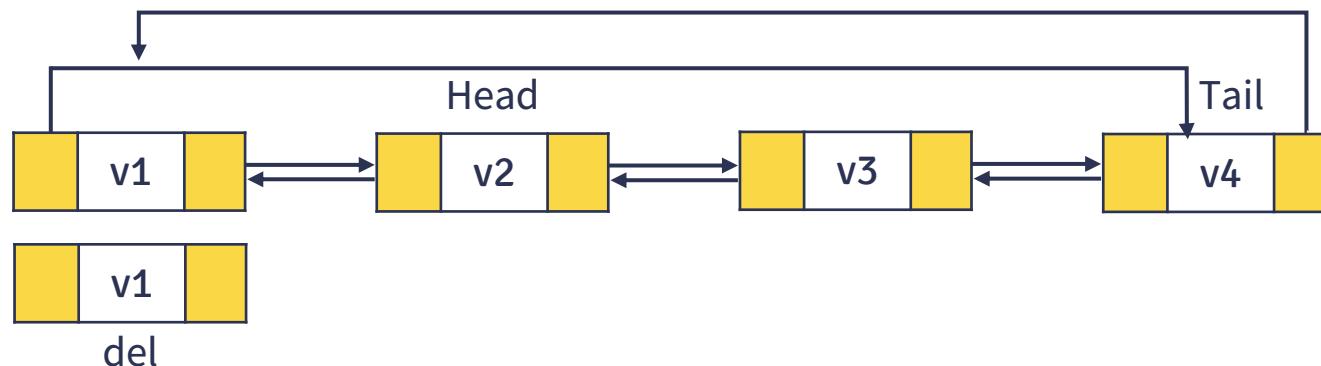
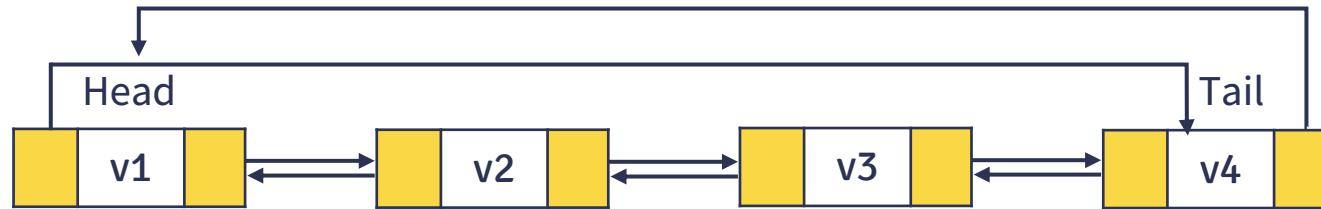
Added at Beginning Node ?



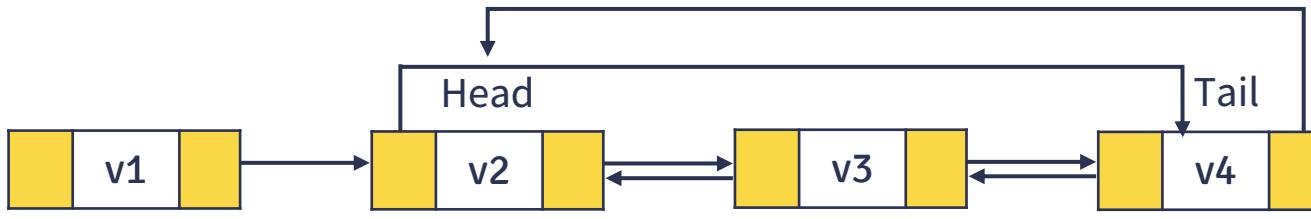
Added at Last Node ?



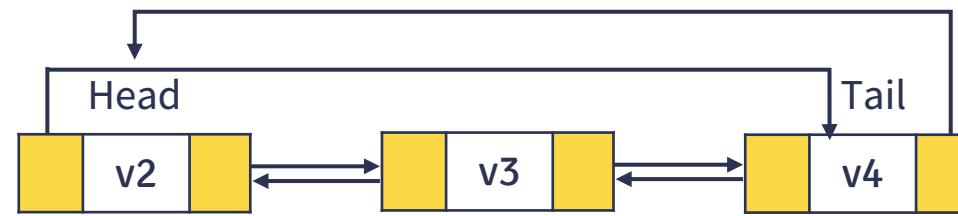
Delete the First Node ?



Delete the First Node ?



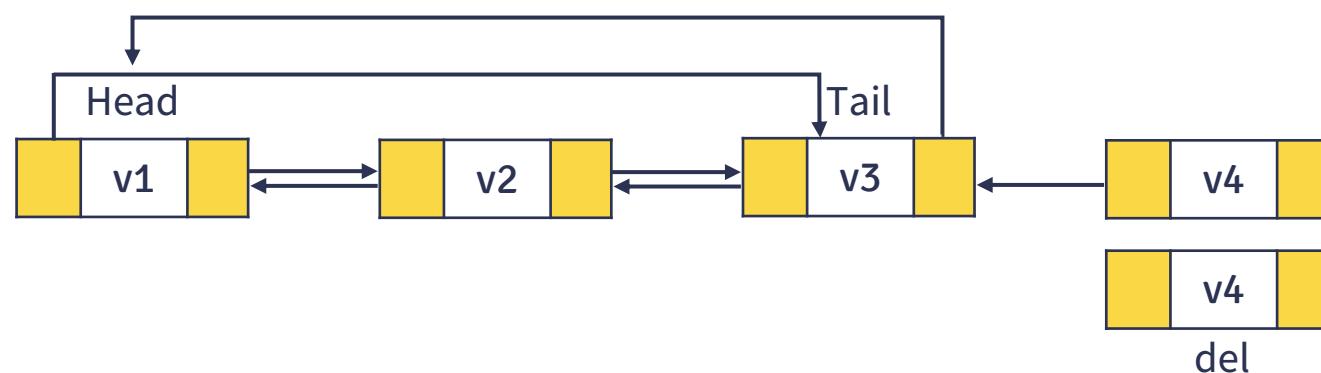
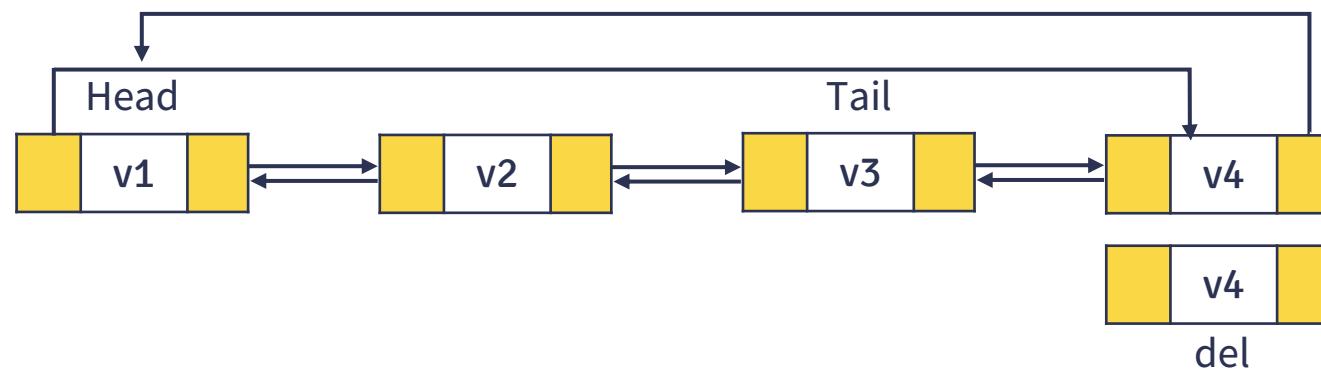
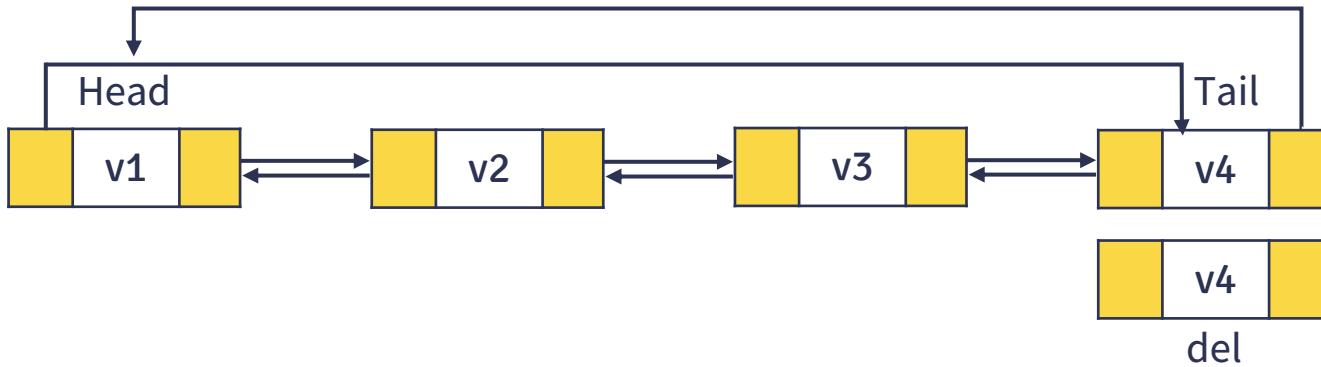
del



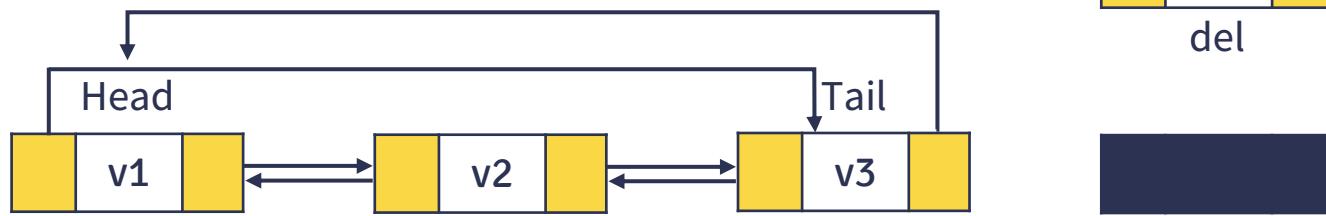
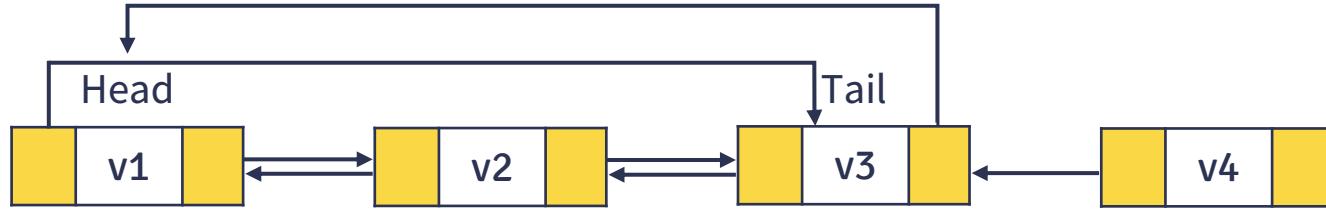
del

del

Delete the Last Node ?



Delete the Last Node ?



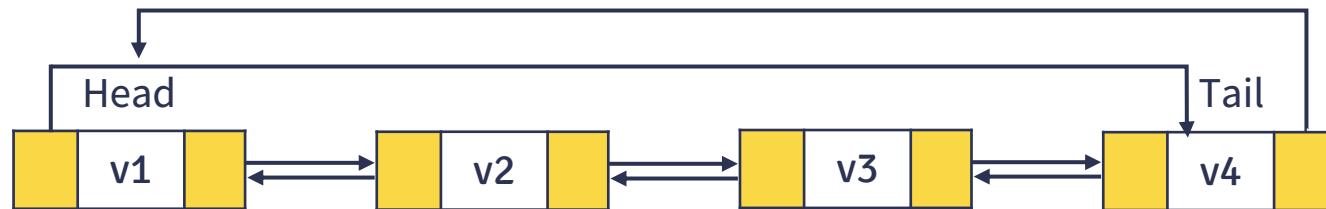
v4
del



del



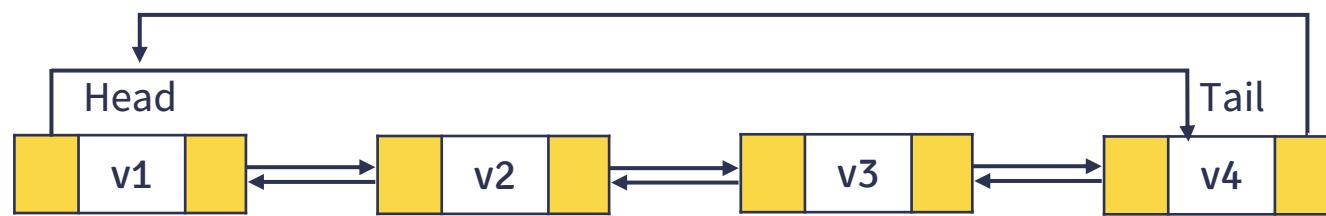
Added at Middle Node ?



Tambah ke posisi 7



newNode



===== loop ($1 < \text{posisi} - 1$) =====

$\text{cur} \rightarrow \text{next}$

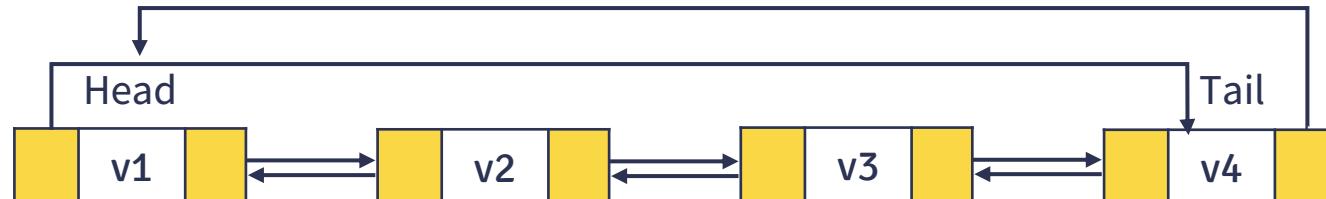


cur

newNode

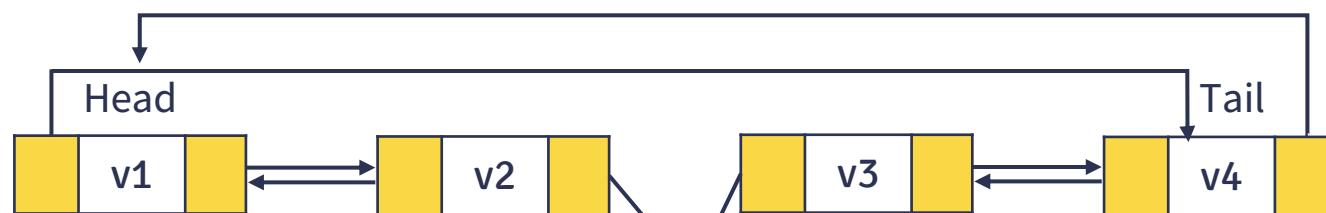
afterNode

Added at Middle Node ?

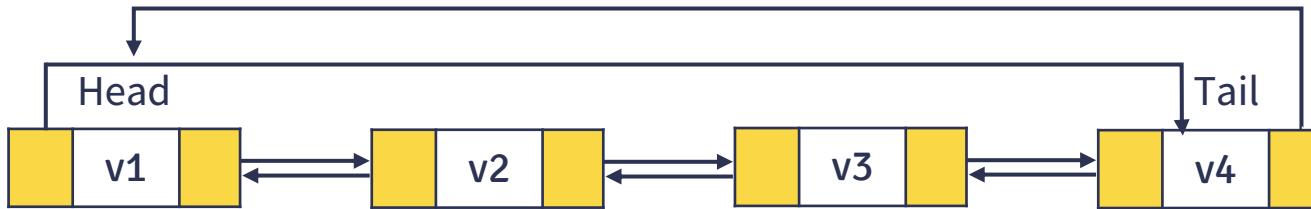


===== loop ($1 < \text{posisi} - 1$) =====>

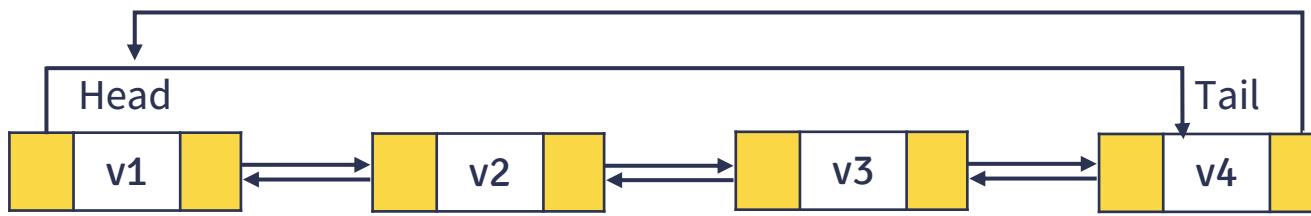
cur->next



Delete at Middle Node ?

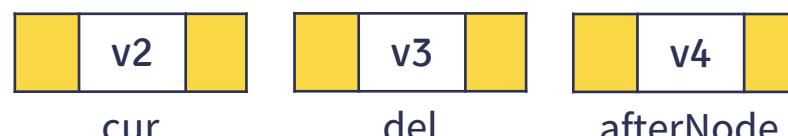


Hapus posisi ke-7



===== loop ($1 < \text{posisi} - 1$) =====>

cur->next del->next

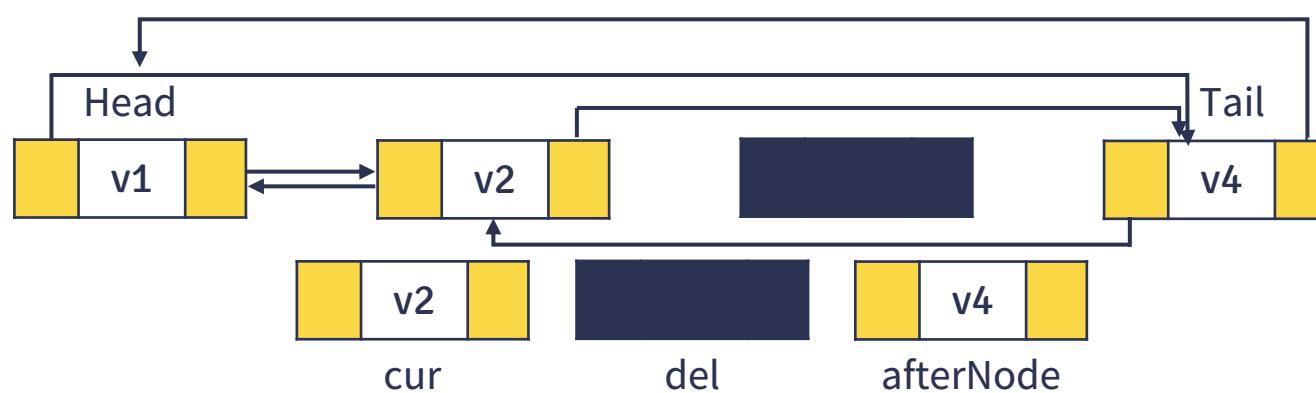
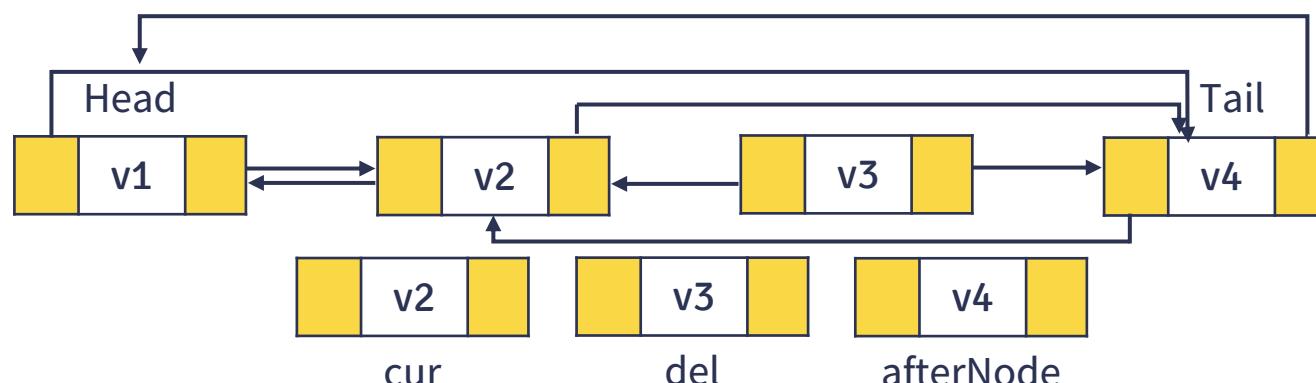
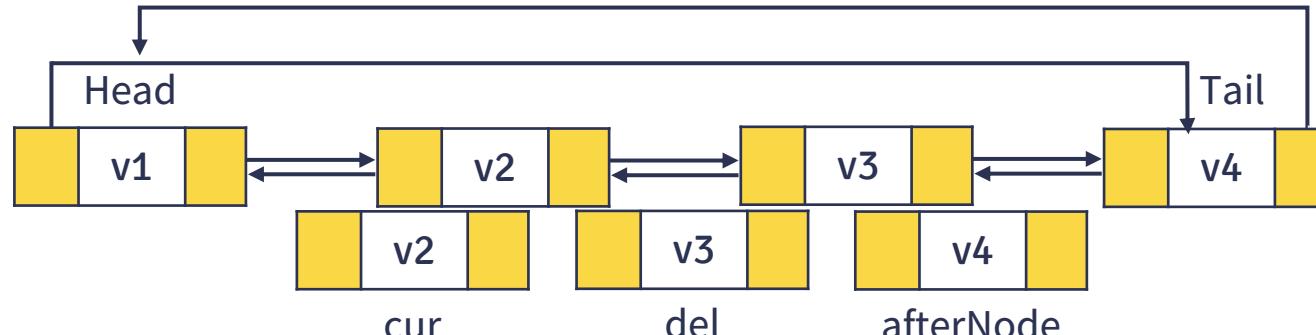


cur

del

afterNode

Delete at Middle Node ?





Video Selanjutnya

Stack (tumpukan)



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 5

Stack (tumpukan)

Apa itu Stack ?

- Stack atau tumpukan adalah sebuah kumpulan data dimana data yang diletakkan di atas data yang lain.
- Stack atau tumpukan adalah kumpulan elemen yang hanya dapat ditambah dan atau dihapus dari satu ujung (gerbang) yang sama.
- Karakteristik penting stack adalah bersifat LIFO (Last In First Out) artinya data yang terakhir masuk merupakan data yang akan keluar terlebih dahulu.
- Stack mempunyai batas maksimal banyak data.
- Stack bisa diimplementasikan dengan array dan linked list.

Analogi Stack ?



Operasi Standar pada Stack ?

- Push

Menambahkan elemen ke atas stack (tumpukan).

- Pop

Menghapus satu data ditumpukan teratas.

- isFull

Memeriksa apakah ruang stack sudah penuh atau belum.

- isEmpty

Memeriksa apakah ruang stack kosong atau tidak.

- Peek

Melihat atau mengintip data diposisi tertentu.

- Count

Menghitung banyak data pada stack.

- Change

Mengubah data diposisi tertentu.

- Display

Mencetak semua data pada stack

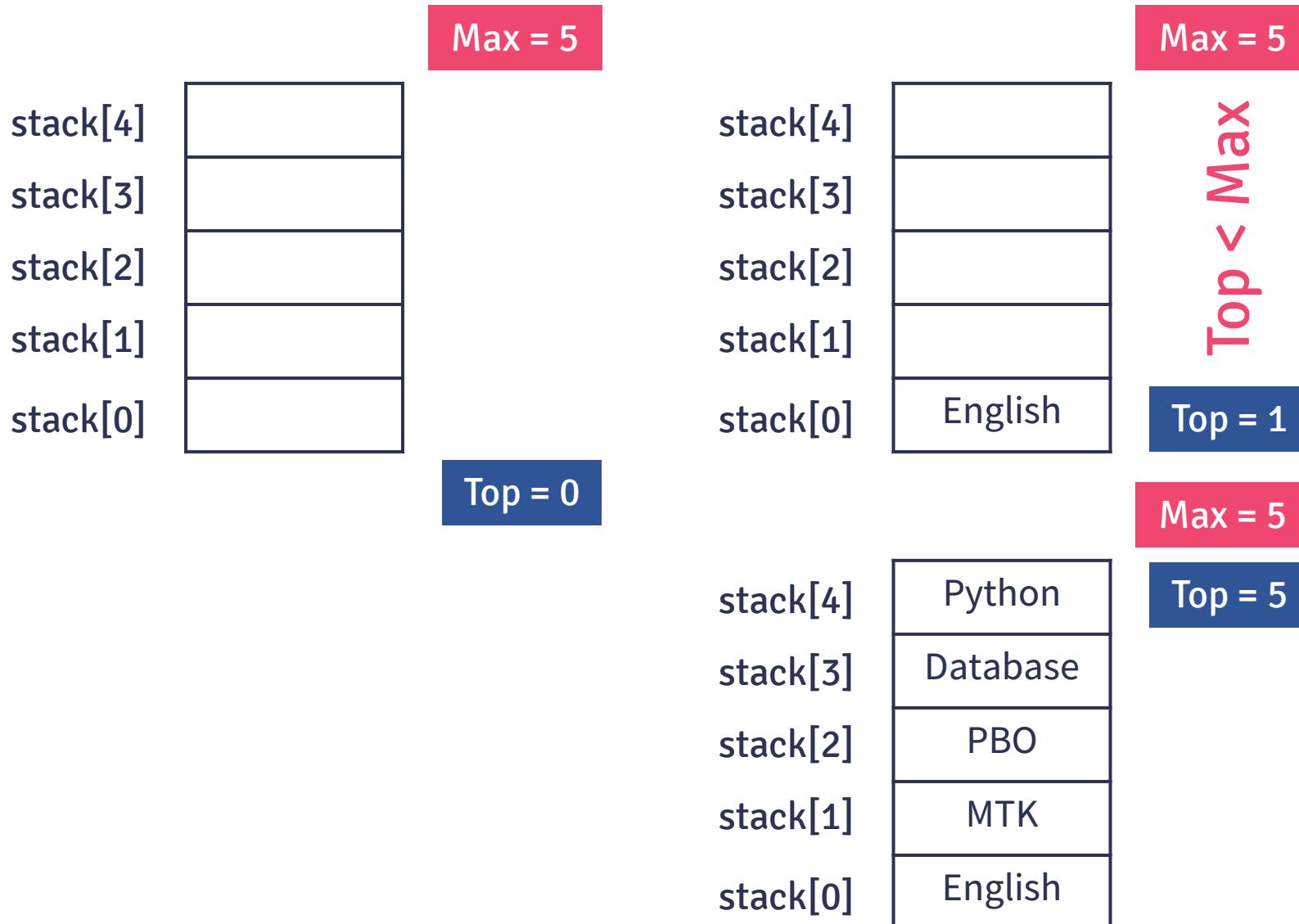
- Destroy

Menghapus atau membersihkan semua data pada stack

Penerapan Stack ?

- Tombol Back dan Next
- Tombol Undo dan Redo
- Pada beberapa literatur menyebutkan bahwa stack umumnya digunakan untuk memisahkan ekspresi aritmatika.
- Penyelesaian pencarian rute labirin.

Konsep Stack pakai Array ?



Konsep Stack pakai Linked List ?

Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Head/Tail



Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Head





Video Selanjutnya

Queue (antrian)





Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 5

Stack (tumpukan)

Apa itu Stack ?

- Stack atau tumpukan adalah sebuah kumpulan data dimana data yang diletakkan di atas data yang lain.
- Stack atau tumpukan adalah kumpulan elemen yang hanya dapat ditambah dan atau dihapus dari satu ujung (gerbang) yang sama.
- Karakteristik penting stack adalah bersifat LIFO (Last In First Out) artinya data yang terakhir masuk merupakan data yang akan keluar terlebih dahulu.
- Stack mempunyai batas maksimal banyak data.
- Stack bisa diimplementasikan dengan array dan linked list.

Analogi Stack ?



Operasi Standar pada Stack ?

- Push

Menambahkan elemen ke atas stack (tumpukan).

- Pop

Menghapus satu data ditumpukan teratas.

- isFull

Memeriksa apakah ruang stack sudah penuh atau belum.

- isEmpty

Memeriksa apakah ruang stack kosong atau tidak.

- Peek

Melihat atau mengintip data diposisi tertentu.

- Count

Menghitung banyak data pada stack.

- Change

Mengubah data diposisi tertentu.

- Display

Mencetak semua data pada stack

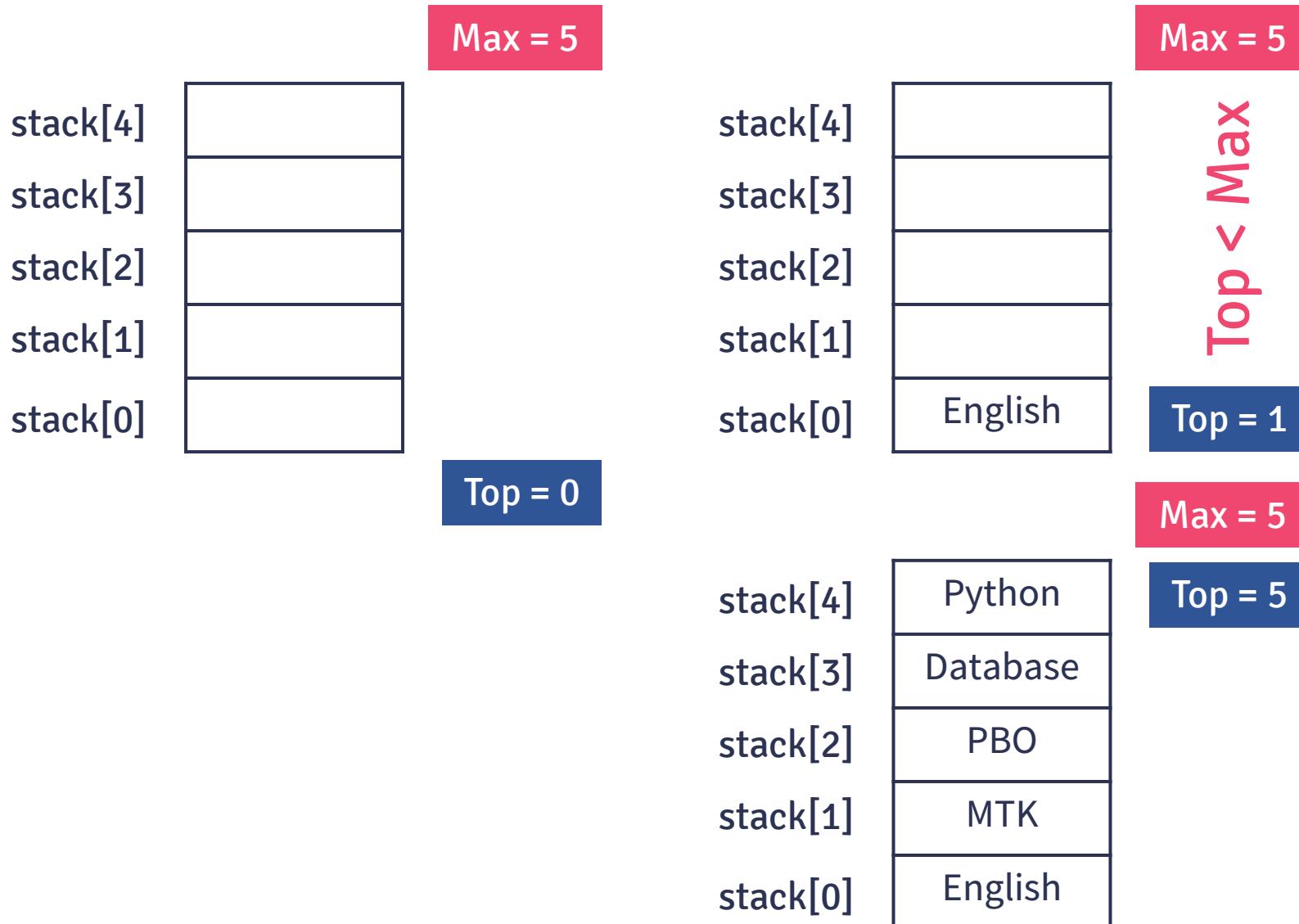
- Destroy

Menghapus atau membersihkan semua data pada stack

Penerapan Stack ?

- Tombol Back dan Next
- Tombol Undo dan Redo
- Pada beberapa literatur menyebutkan bahwa stack umumnya digunakan untuk memisahkan ekspresi aritmatika.
- Penyelesaian pencarian rute labirin.

Konsep Stack pakai Array ?



Konsep Stack pakai Linked List ?

Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Head/Tail



Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Head





Video Selanjutnya

Queue (antrian)



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 6

Queue (antrian)

Apa itu Queue ?

- Queue (Antrian) adalah suatu kumpulan data yang mana penambahan data / elemen hanya dapat dilakukan pada sisi belakang sedangkan penghapusan / pengeluaran elemen dilakukan pada sisi depan.
- Karakteristik penting queue adalah bersifat FIFO (First In First Out) artinya data yang terlebih dahulu masuk merupakan data yang akan keluar terlebih dahulu.
- Seperti halnya pada antrian yang biasa kita lakukan sehari-hari, di manapun. Antrian dimulai dari depan ke belakang, jika yang berada didepan belum pergi meninggalkan antrian maka antrian yang ada dibelakang akan terus bertambah dan antrian paling belakang disini dinamakan rear/tail.
- Jika ada yang keluar dari antrian (dequeue) maka data tersebut adalah yang paling depan (head/front), dan data berikutnya setelah data yang keluar berubah menjadi yang paling depan (head/front).

Analogi Queue ?



designed by  freepik

Operasi Standar pada Queue ?

- Enqueue

Menambahkan elemen ke antrian.

- Dequeue

Menghapus data pertama pada antrian.

- isFull

Memeriksa apakah ruang queue sudah penuh atau belum.

- isEmpty

Memeriksa apakah ruang queue kosong atau tidak.

- Count

Menghitung banyak data pada antrian.

- Display / View

Mencetak semua data pada antrian

- Destroy / Clear

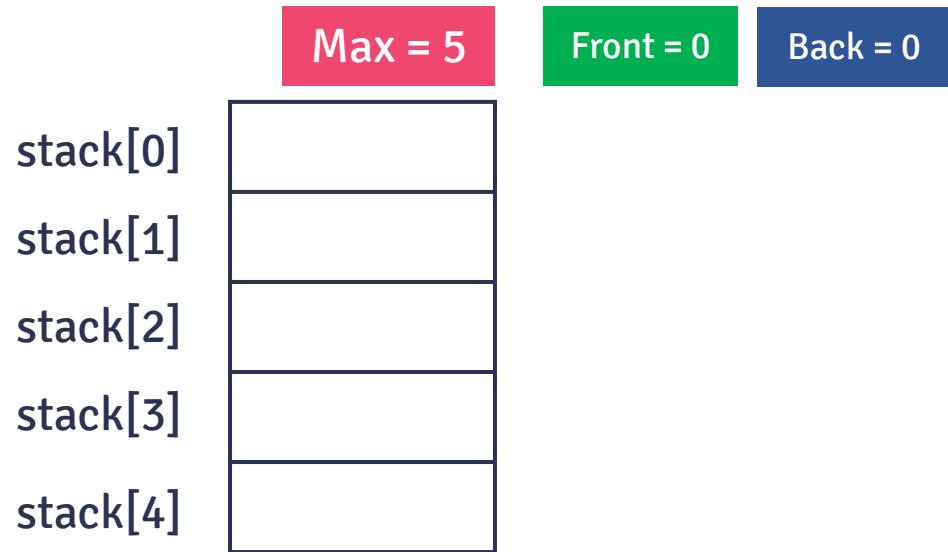
Menghapus atau membersihkan semua data pada queue

Penerapan Queue ?

- Representasi antrian dikehidupan nyata & antrian pada memori.

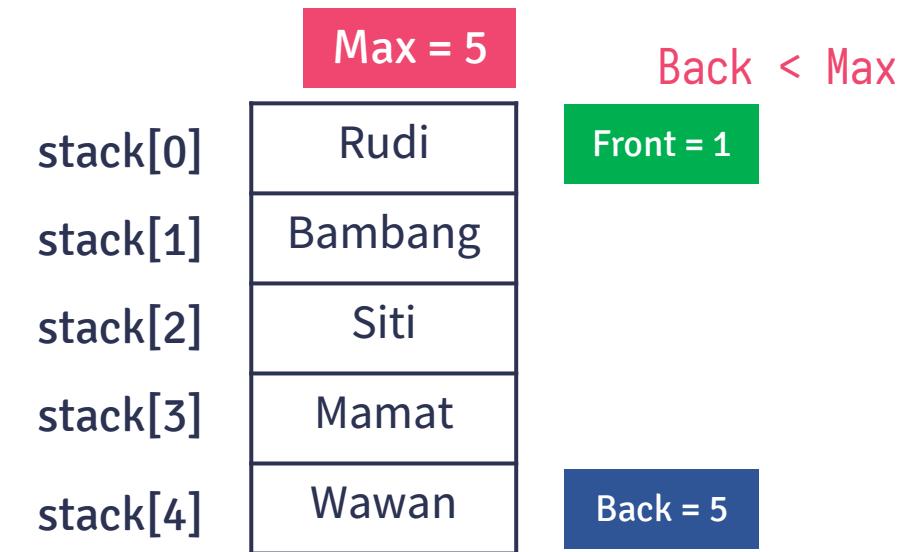
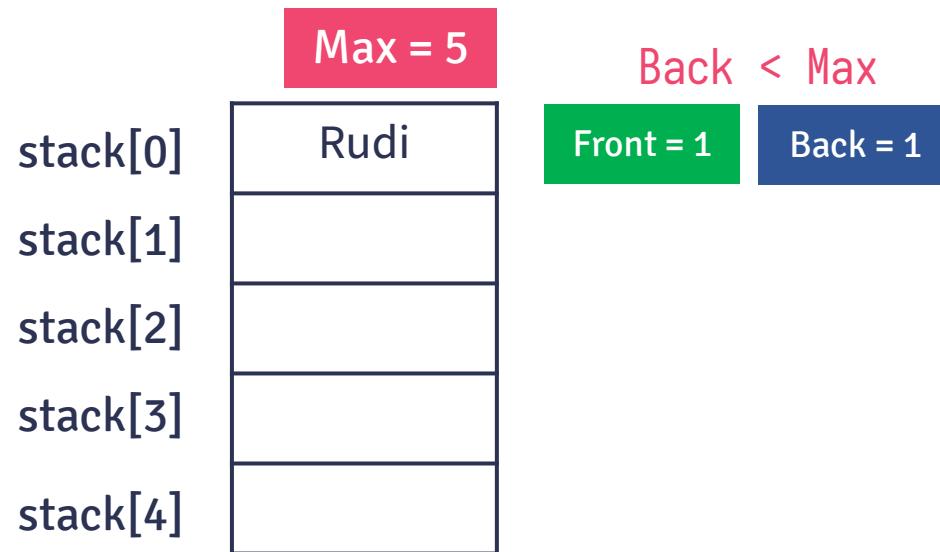


Konsep Queue pakai Array ?



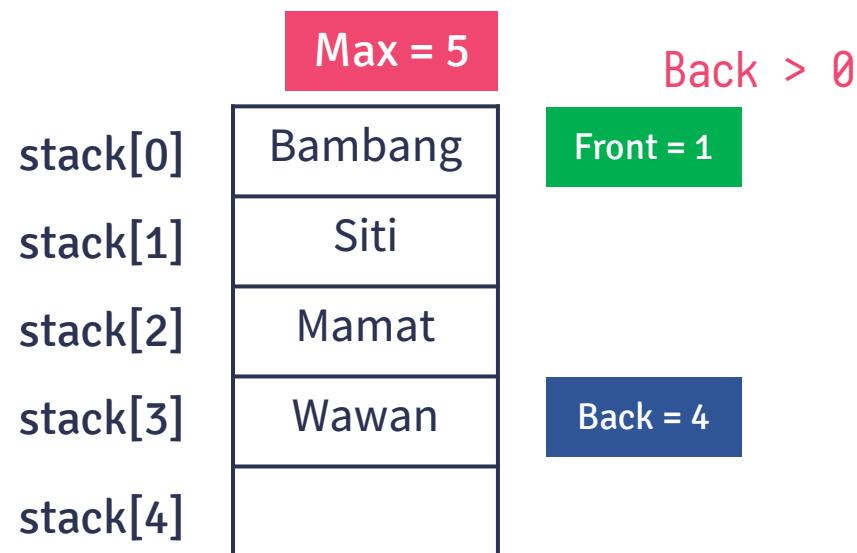
Konsep Queue pakai Array ?

Penambahan Data



Konsep Queue pakai Array ?

Penghapusan Data



Konsep Queue pakai Linked List ?

Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Head/Tail



Head





Video Selanjutnya

Hashing



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 6

Queue (antrian)

Apa itu Queue ?

- Queue (Antrian) adalah suatu kumpulan data yang mana penambahan data / elemen hanya dapat dilakukan pada sisi belakang sedangkan penghapusan / pengeluaran elemen dilakukan pada sisi depan.
- Karakteristik penting queue adalah bersifat FIFO (First In First Out) artinya data yang terlebih dahulu masuk merupakan data yang akan keluar terlebih dahulu.
- Seperti halnya pada antrian yang biasa kita lakukan sehari-hari, di manapun. Antrian dimulai dari depan ke belakang, jika yang berada didepan belum pergi meninggalkan antrian maka antrian yang ada dibelakang akan terus bertambah dan antrian paling belakang disini dinamakan rear/tail.
- Jika ada yang keluar dari antrian (dequeue) maka data tersebut adalah yang paling depan (head/front), dan data berikutnya setelah data yang keluar berubah menjadi yang paling depan (head/front).

Analogi Queue ?



designed by  freepik

Operasi Standar pada Queue ?

- Enqueue

Menambahkan elemen ke antrian.

- Dequeue

Menghapus data pertama pada antrian.

- isFull

Memeriksa apakah ruang queue sudah penuh atau belum.

- isEmpty

Memeriksa apakah ruang queue kosong atau tidak.

- Count

Menghitung banyak data pada antrian.

- Display / View

Mencetak semua data pada antrian

- Destroy / Clear

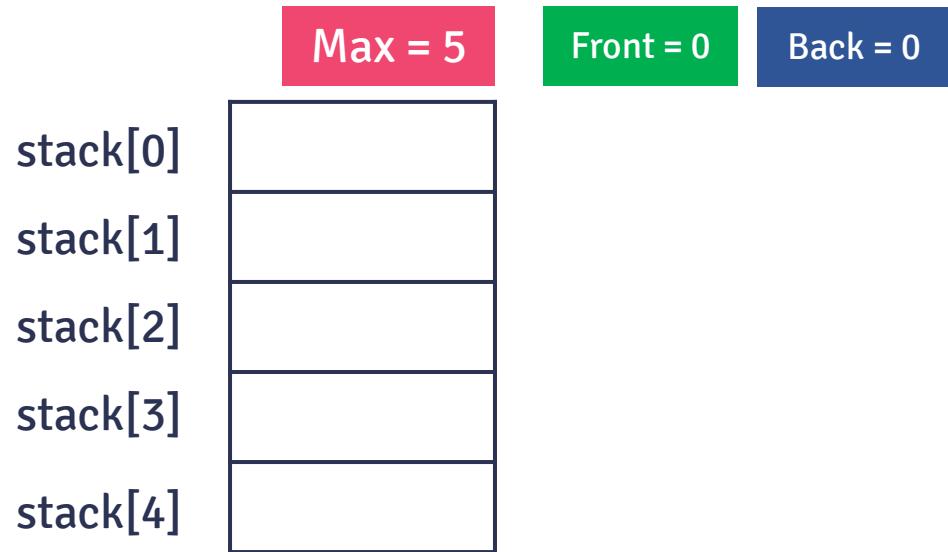
Menghapus atau membersihkan semua data pada queue

Penerapan Queue ?

- Representasi antrian dikehidupan nyata & antrian pada memori.

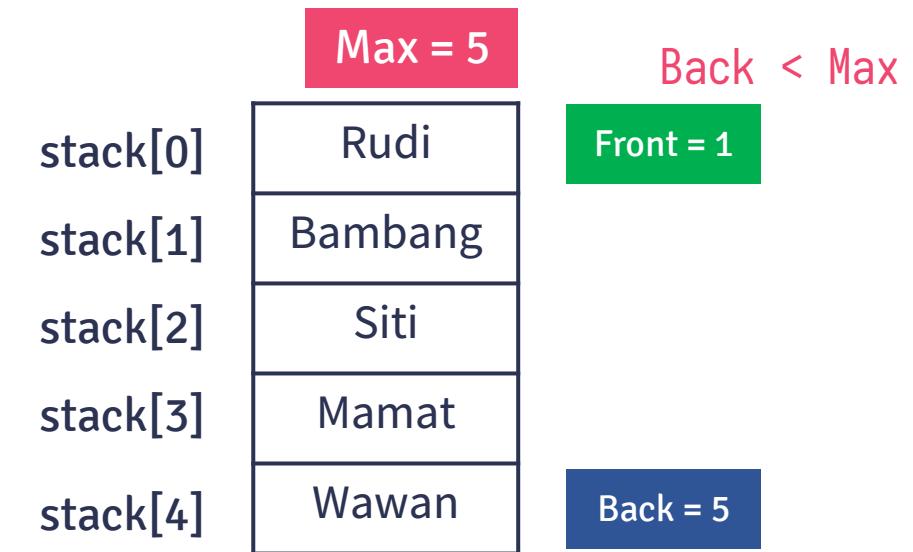
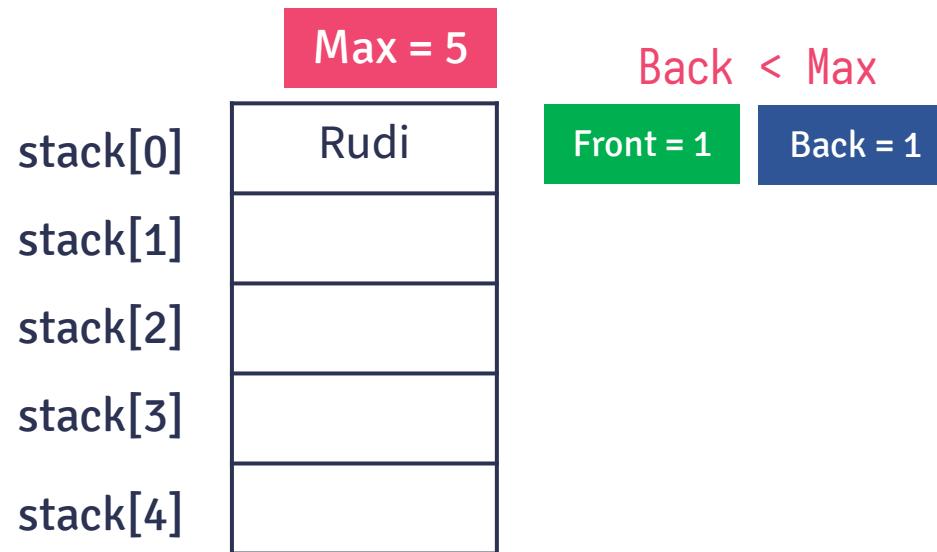


Konsep Queue pakai Array ?



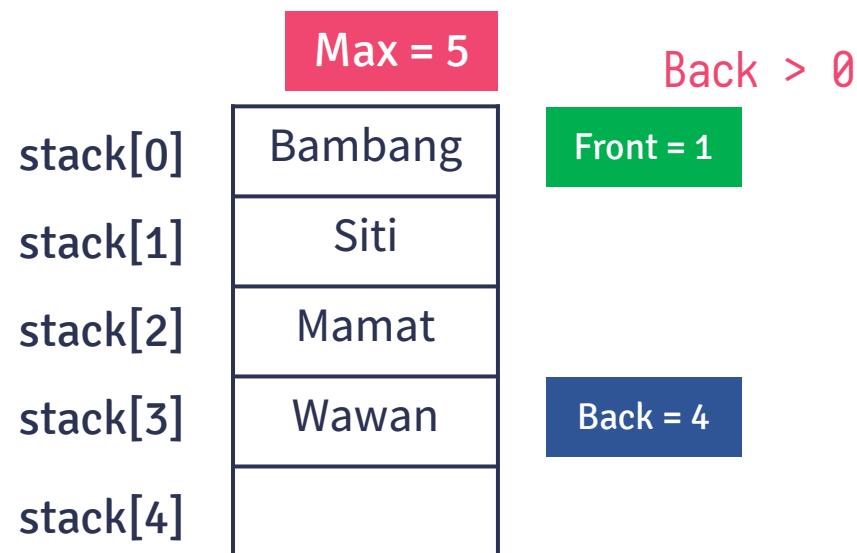
Konsep Queue pakai Array ?

Penambahan Data



Konsep Queue pakai Array ?

Penghapusan Data



Konsep Queue pakai Linked List ?

Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Count() Max = 4

```
if Count() == Max  
    DISPLAY("FULL")  
else  
    if Count() == 0  
        Create(data)  
    else  
        push(data)  
    endif  
endif
```

Head/Tail



Head





Video Selanjutnya

Hashing



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 7

Hashing

Istilah yang Perlu Diketahui ?

- Hashing
- Hash Table
- Hash Function
- Collision
- Collision Resolution



Hashing ?

- Hashing adalah transformasi aritmatika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya.
- Menurut bahasanya, hash berarti memenggal dan kemudian menggabungkan.
- Hashing digunakan sebagai metode untuk menyimpan data dalam sebuah larik (array) agar penyimpanan data, pencarian data, penambahan data, dan penghapusan data dapat dilakukan dengan cepat.
- Hashing digunakan untuk mengindex dan mendapatkan kembali key di database (hash table), karena lebih cepat untuk mengambil key yang sudah dihash daripada mencarinya menggunakan original value.
- Hashing juga dikenal sebagai konsep pendistribusian key dalam sebuah array (Hash Table) menggunakan fungsi yang sudah diketahui sebelumnya (Hash Function).

Hash Table ?

- Hash Table adalah array dengan sel-sel yang ukurannya telah ditentukan dan dapat berisi data atau key yang berkesesuaian dengan data.
- Hash Table adalah sebuah array untuk menyimpan original string. Index dari hash table adalah hashed key.
- Besarnya hash table lebih kecil dari jumlah key yang ada, sehingga memungkinkan ada string yang memiliki hashed key yang sama.

Hash Function ?

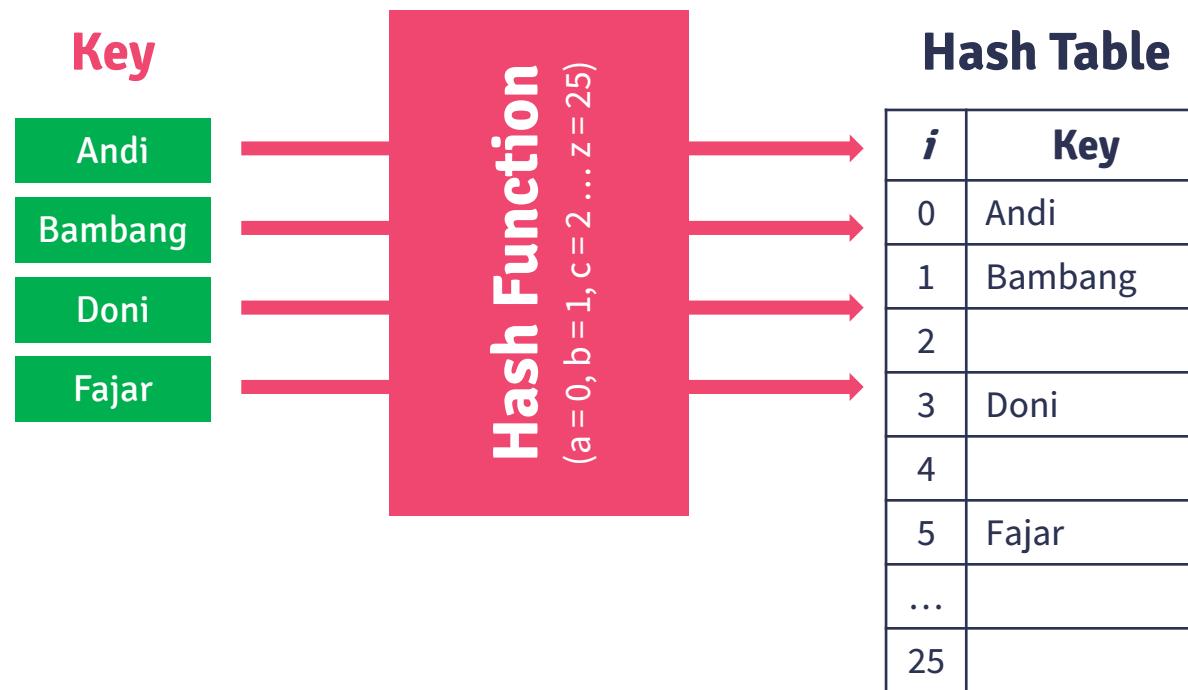
- Hash Function adalah cara yang kita buat untuk men-transformasi aritmetika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya.
- Secara teori, kompleksitas waktu ($T(n)$) dari fungsi hash yang ideal adalah algoritma konstan $O(1)$. Untuk mencapai itu setiap record membutuhkan suatu kunci yang unik.

Notes ?

- Salah satu kegunaan hashing adalah untuk mengamankan tulisan yang biasanya diterapkan untuk menyembunyikan password asli.
- Teknik dan cara hashing semakin berkembang seiring dengan keamanan data yang terus bisa diretas.

Cara-cara Hash Function ?

- Untuk dasarnya, biasanya mengambil dari satu karakter awal pada string original dan diubah menjadi angka index yang nantinya disimpan ke hash table.



Cara-cara Hash Function ?

- Banyak cara untuk melakukan hash sebuah string menjadi hashed key. Berikut adalah cara-cara penting untuk membuat hash function :

- **Mid-Square**
- **Division (Most Common)**
- **Folding**
- **Digit Extraction**
- **Rotating Hash**
- **Truncation**
- dll..

Mid-Square ?

- Pangkatkan key, dan ambil bit pada bagian tengah dari hasil pangkat untuk dijadikan hash-key.
- Jika key berupa string maka ubah menjadi number dengan cara :

U D I N

$$21 + 4 + 9 + 14$$

$$48^2 = 2304$$

30

A D I

$$1 + 4 + 9$$

$$14^2 = 196 = 0196$$

19

Division ?

- Lakukan pembagian pada key dengan operator modulus (sesuai dengan jumlah hash table).
- Jika key berupa string maka ubah menjadi number dengan cara :

U D I N

21 + 4 + 9 + 14

48 % 20

8

A D I

1 + 4 + 9

14 % 20

14

Folding ?

- Hash Function folding akan men-transformasi string ke hashed key dengan 2 Langkah :

1. Bagi key menjadi bagian-bagian yang memiliki jumlah digit sama.
2. Jumlahkan setiap bagian.

U	D	I	N
21	4	9	14

$$21 + 49 + 14$$

84

O	K	I
15	11	9

$$01 + 51 + 19$$

71

O	K	I
15	11	9

$$51 + 19$$

70



Folding ?

- Hash Function folding akan men-transformasi string ke hashed key dengan 2 Langkah :

1. Bagi key menjadi bagian-bagian yang memiliki jumlah digit sama.
2. Jumlahkan setiap bagian.

U	D	I	N
21	4	9	14

$$41 + 94 + 12$$

147

O	K	I
15	11	9

$$91 + 15$$

106

Digit Extraction ?

- Mendapatkan hashed key dengan cara mengambil digit-digit tertentu dari sebuah key .

U D I N
21 4 9 14

241

O K I
15 11 9

119

Rotating Hash ?

- Mendapatkan hashed key dengan cara membalik urutan dari key.

U D I N
21 4 9 14
4 1 9 4 2 1

O K I
15 11 9
9 1 1 5 1

Truncation ?

- Mendapatkan hashed key dengan cara memenggal key sebanyak k (tidak boleh dilewati) dari n digit, dimana $k < n$.

U	D	I	N
21	4	9	14
2 1 4			

U	D	I	N
21	4	9	14
1 4 9			

U	D	I	N
21	4	9	14
4 9 1 4			

Hash Function yang Baik ?

- Hash Function harus memiliki sifat berikut :
 - Mudah dihitung
 - Dua key yang berbeda akan dipetakan pada dua sel yang berbeda pada array. Tapi secara umum tidak bisa berlaku karena bisa jadi dua key yang berbeda mempunyai hasil hashed key yang sama (tabrakan).
 - Membagi key secara rata pada seluruh sel.
- Sebuah hash function sederhana adalah menggunakan fungsi modulus (sisa bagi) dengan bilangan prima.
- Dapat menggunakan manipulasi digit dengan kompleksitas rendah dan distribusi key yang rata.

Memilih Hash Function ?

- Sebuah hash function yang bagus memiliki dua kriteria :
 - Mudah dan cepat dihitung
 - Harus meminimalkan juga collisions (tabrakan) yang terjadi.

Collision ?

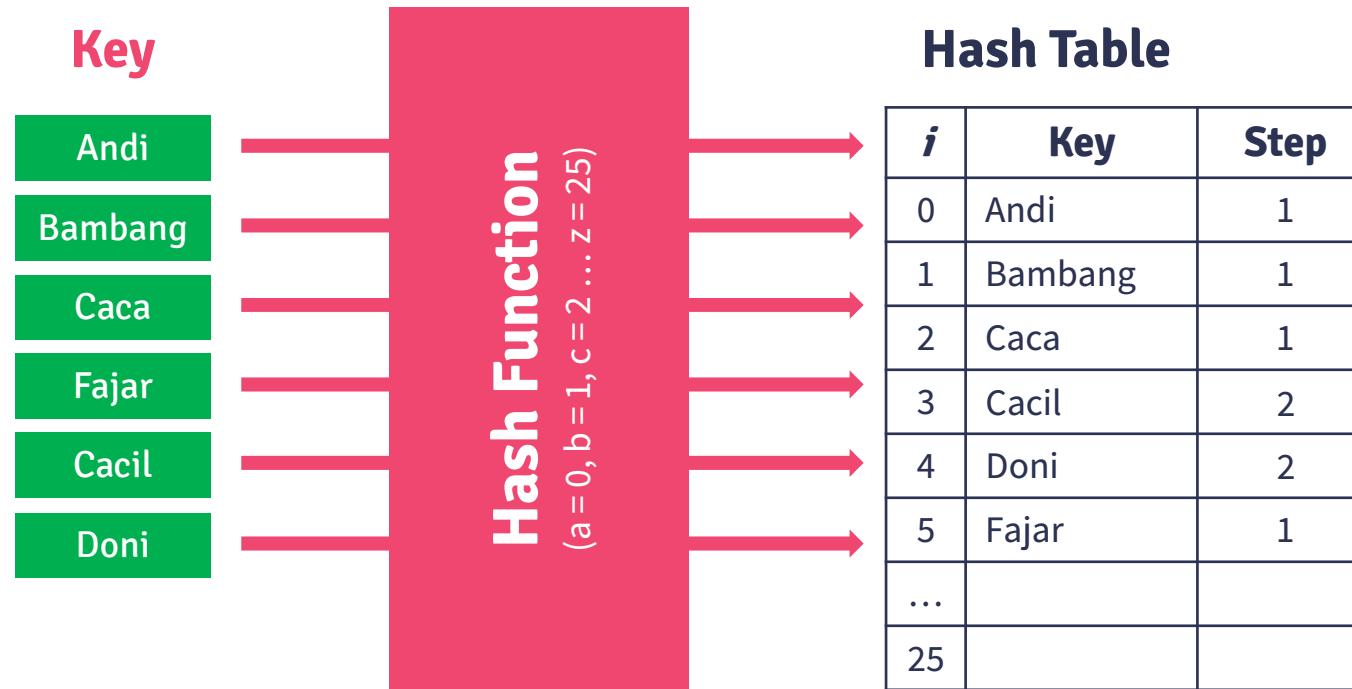
- Dikatakan terjadi collision (tabrakan) jika dua buah keys dipetakkan pada sebuah sel yang sama.
- Collision bisa terjadi saat melakukan insertion (penambahan data).
- Penyelesaian bila terjadi collision (tabrakan) disebut *Collision Resolution*.
- Dibutuhkan prosedur tambahan untuk mengatasi terjadinya collision (tabrakan).

Collision Resolution ?

- Ada dua strategi umum dalam melakukan Collision Resolution :
 - **Closed Hashing (Open Addressing)**
 - Linear Probing.
 - Quadratic Probing.
 - Double Hashing.
 - **Open Hashing (Chaining)**

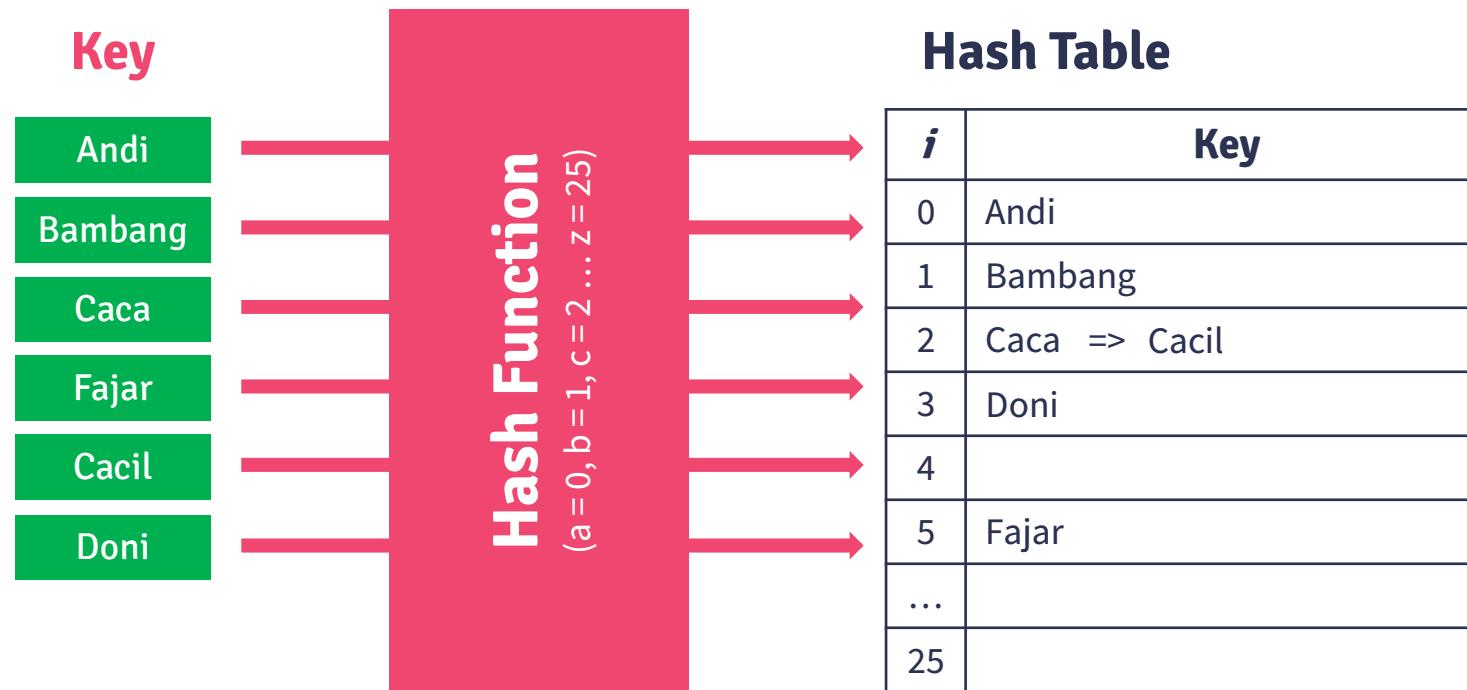
Closed Hashing – Linear Probing ?

- Ideanya adalah mencari alternatif sel lain pada hash table.



Open Hashing (Chaining) ?

- Ideanya meletakkan key pada table menggunakan linked list.





Video Selanjutnya

**Ngoding Program Login Sederhana
Dengan Hashing C++**



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 7

Hashing

Istilah yang Perlu Diketahui ?

- Hashing
- Hash Table
- Hash Function
- Collision
- Collision Resolution



Hashing ?

- Hashing adalah transformasi aritmatika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya.
- Menurut bahasanya, hash berarti memenggal dan kemudian menggabungkan.
- Hashing digunakan sebagai metode untuk menyimpan data dalam sebuah larik (array) agar penyimpanan data, pencarian data, penambahan data, dan penghapusan data dapat dilakukan dengan cepat.
- Hashing digunakan untuk mengindex dan mendapatkan kembali key di database (hash table), karena lebih cepat untuk mengambil key yang sudah dihash daripada mencarinya menggunakan original value.
- Hashing juga dikenal sebagai konsep pendistribusian key dalam sebuah array (Hash Table) menggunakan fungsi yang sudah diketahui sebelumnya (Hash Function).

Hash Table ?

- Hash Table adalah array dengan sel-sel yang ukurannya telah ditentukan dan dapat berisi data atau key yang berkesesuaian dengan data.
- Hash Table adalah sebuah array untuk menyimpan original string. Index dari hash table adalah hashed key.
- Besarnya hash table lebih kecil dari jumlah key yang ada, sehingga memungkinkan ada string yang memiliki hashed key yang sama.

Hash Function ?

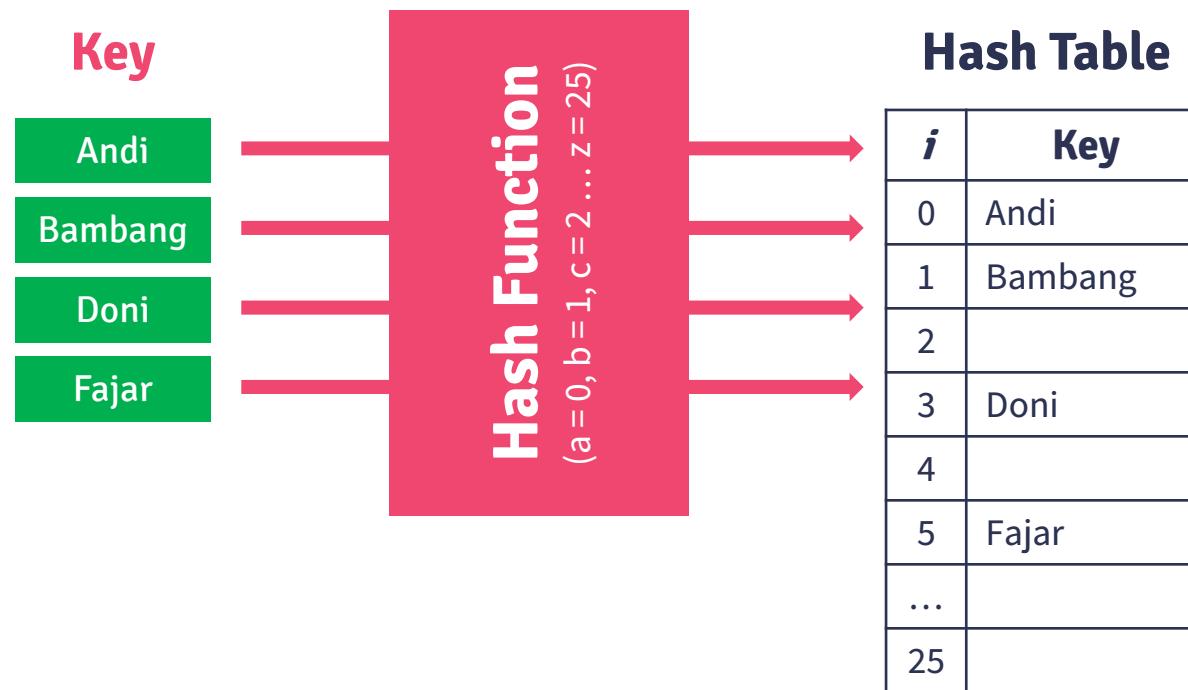
- Hash Function adalah cara yang kita buat untuk men-transformasi aritmetika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya.
- Secara teori, kompleksitas waktu ($T(n)$) dari fungsi hash yang ideal adalah algoritma konstan $O(1)$. Untuk mencapai itu setiap record membutuhkan suatu kunci yang unik.

Notes ?

- Salah satu kegunaan hashing adalah untuk mengamankan tulisan yang biasanya diterapkan untuk menyembunyikan password asli.
- Teknik dan cara hashing semakin berkembang seiring dengan keamanan data yang terus bisa diretas.

Cara-cara Hash Function ?

- Untuk dasarnya, biasanya mengambil dari satu karakter awal pada string original dan diubah menjadi angka index yang nantinya disimpan ke hash table.



Cara-cara Hash Function ?

- Banyak cara untuk melakukan hash sebuah string menjadi hashed key. Berikut adalah cara-cara penting untuk membuat hash function :

- **Mid-Square**
- **Division (Most Common)**
- **Folding**
- **Digit Extraction**
- **Rotating Hash**
- **Truncation**
- dll..

Mid-Square ?

- Pangkatkan key, dan ambil bit pada bagian tengah dari hasil pangkat untuk dijadikan hash-key.
- Jika key berupa string maka ubah menjadi number dengan cara :

U D I N

21 + 4 + 9 + 14

$48^2 = 2304$

30

A D I

1 + 4 + 9

$14^2 = 196 = 0196$

19

Division ?

- Lakukan pembagian pada key dengan operator modulus (sesuai dengan jumlah hash table).
- Jika key berupa string maka ubah menjadi number dengan cara :

U D I N

21 + 4 + 9 + 14

48 % 20

8

A D I

1 + 4 + 9

14 % 20

14

Folding ?

- Hash Function folding akan men-transformasi string ke hashed key dengan 2 Langkah :

1. Bagi key menjadi bagian-bagian yang memiliki jumlah digit sama.
2. Jumlahkan setiap bagian.

U	D	I	N
21	4	9	14

$$21 + 49 + 14$$

84

O	K	I
15	11	9

$$01 + 51 + 19$$

71

O	K	I
15	11	9

$$51 + 19$$

70



Folding ?

- Hash Function folding akan men-transformasi string ke hashed key dengan 2 Langkah :

1. Bagi key menjadi bagian-bagian yang memiliki jumlah digit sama.
2. Jumlahkan setiap bagian.

U	D	I	N
21	4	9	14

$$41 + 94 + 12$$

147

O	K	I
15	11	9

$$91 + 15$$

106

Digit Extraction ?

- Mendapatkan hashed key dengan cara mengambil digit-digit tertentu dari sebuah key .

U D I N
21 4 9 14

241

O K I
15 11 9

119

Rotating Hash ?

- Mendapatkan hashed key dengan cara membalik urutan dari key.

U D I N
21 4 9 14
4 1 9 4 2 1

O K I
15 11 9
9 1 1 5 1

Truncation ?

- Mendapatkan hashed key dengan cara memenggal key sebanyak k (tidak boleh dilewati) dari n digit, dimana $k < n$.

U	D	I	N
21	4	9	14
2 1 4			

U	D	I	N
21	4	9	14
1 4 9			

U	D	I	N
21	4	9	14
4 9 1 4			

Hash Function yang Baik ?

- Hash Function harus memiliki sifat berikut :
 - Mudah dihitung
 - Dua key yang berbeda akan dipetakan pada dua sel yang berbeda pada array. Tapi secara umum tidak bisa berlaku karena bisa jadi dua key yang berbeda mempunyai hasil hashed key yang sama (tabrakan).
 - Membagi key secara rata pada seluruh sel.
- Sebuah hash function sederhana adalah menggunakan fungsi modulus (sisa bagi) dengan bilangan prima.
- Dapat menggunakan manipulasi digit dengan kompleksitas rendah dan distribusi key yang rata.

Memilih Hash Function ?

- Sebuah hash function yang bagus memiliki dua kriteria :
 - Mudah dan cepat dihitung
 - Harus meminimalkan juga collisions (tabrakan) yang terjadi.

Collision ?

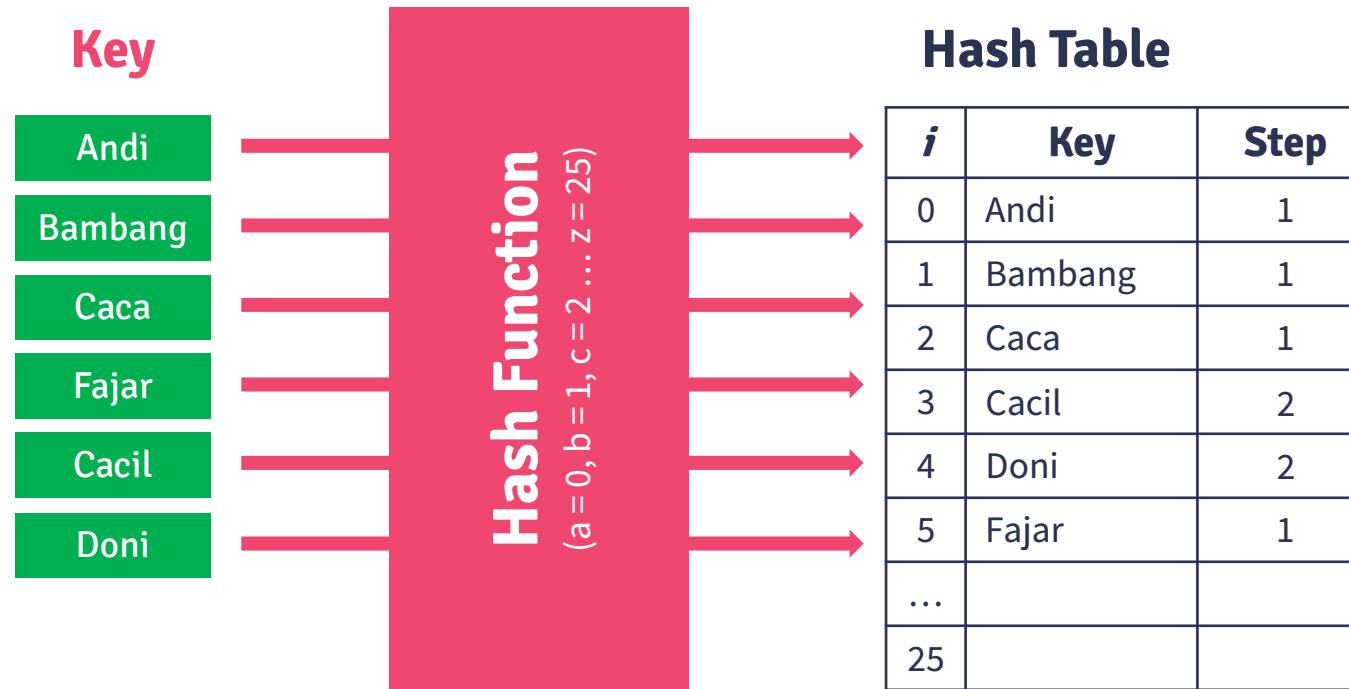
- Dikatakan terjadi collision (tabrakan) jika dua buah keys dipetakkan pada sebuah sel yang sama.
- Collision bisa terjadi saat melakukan insertion (penambahan data).
- Penyelesaian bila terjadi collision (tabrakan) disebut *Collision Resolution*.
- Dibutuhkan prosedur tambahan untuk mengatasi terjadinya collision (tabrakan).

Collision Resolution ?

- Ada dua strategi umum dalam melakukan Collision Resolution :
 - **Closed Hashing (Open Addressing)**
 - Linear Probing.
 - Quadratic Probing.
 - Double Hashing.
 - **Open Hashing (Chaining)**

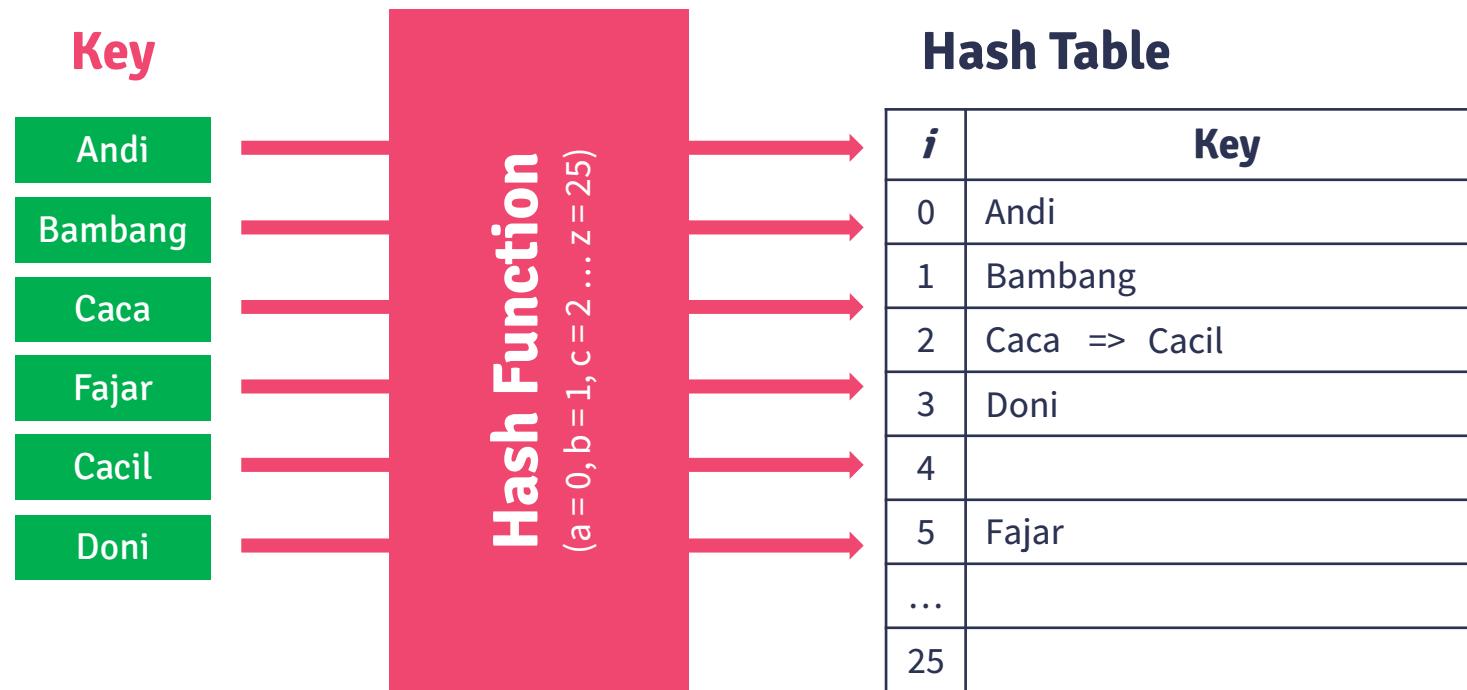
Closed Hashing – Linear Probing ?

- Ideanya adalah mencari alternatif sel lain pada hash table.



Open Hashing (Chaining) ?

- Ideanya meletakkan key pada table menggunakan linked list.





Video Selanjutnya

**Ngoding Program Login Sederhana
Dengan Hashing C++**



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 7

Hashing

Istilah yang Perlu Diketahui ?

- Hashing
- Hash Table
- Hash Function
- Collision
- Collision Resolution



Hashing ?

- Hashing adalah transformasi aritmatika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya.
- Menurut bahasanya, hash berarti memenggal dan kemudian menggabungkan.
- Hashing digunakan sebagai metode untuk menyimpan data dalam sebuah larik (array) agar penyimpanan data, pencarian data, penambahan data, dan penghapusan data dapat dilakukan dengan cepat.
- Hashing digunakan untuk mengindex dan mendapatkan kembali key di database (hash table), karena lebih cepat untuk mengambil key yang sudah dihash daripada mencarinya menggunakan original value.
- Hashing juga dikenal sebagai konsep pendistribusian key dalam sebuah array (Hash Table) menggunakan fungsi yang sudah diketahui sebelumnya (Hash Function).

Hash Table ?

- Hash Table adalah array dengan sel-sel yang ukurannya telah ditentukan dan dapat berisi data atau key yang berkesesuaian dengan data.
- Hash Table adalah sebuah array untuk menyimpan original string. Index dari hash table adalah hashed key.
- Besarnya hash table lebih kecil dari jumlah key yang ada, sehingga memungkinkan ada string yang memiliki hashed key yang sama.

Hash Function ?

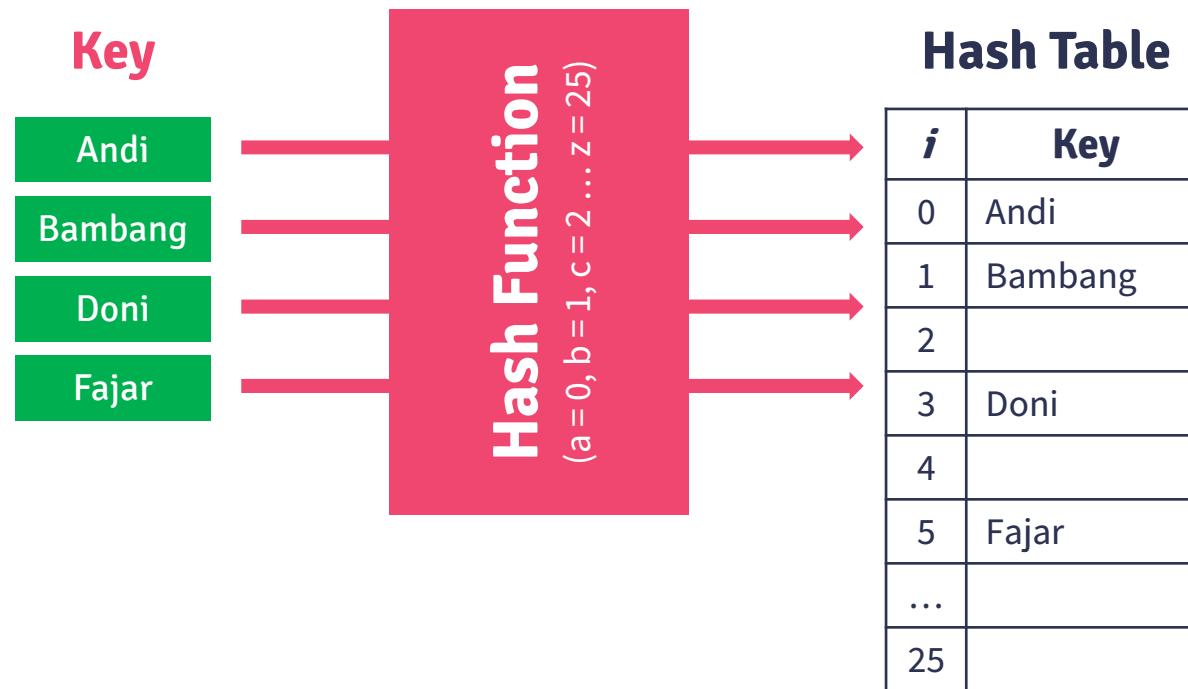
- Hash Function adalah cara yang kita buat untuk men-transformasi aritmetika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya.
- Secara teori, kompleksitas waktu ($T(n)$) dari fungsi hash yang ideal adalah algoritma konstan $O(1)$. Untuk mencapai itu setiap record membutuhkan suatu kunci yang unik.

Notes ?

- Salah satu kegunaan hashing adalah untuk mengamankan tulisan yang biasanya diterapkan untuk menyembunyikan password asli.
- Teknik dan cara hashing semakin berkembang seiring dengan keamanan data yang terus bisa diretas.

Cara-cara Hash Function ?

- Untuk dasarnya, biasanya mengambil dari satu karakter awal pada string original dan diubah menjadi angka index yang nantinya disimpan ke hash table.



Cara-cara Hash Function ?

- Banyak cara untuk melakukan hash sebuah string menjadi hashed key. Berikut adalah cara-cara penting untuk membuat hash function :

- **Mid-Square**
- **Division (Most Common)**
- **Folding**
- **Digit Extraction**
- **Rotating Hash**
- **Truncation**
- dll..

Mid-Square ?

- Pangkatkan key, dan ambil bit pada bagian tengah dari hasil pangkat untuk dijadikan hash-key.
- Jika key berupa string maka ubah menjadi number dengan cara :

U D I N

21 + 4 + 9 + 14

$48^2 = 2304$

30

A D I

1 + 4 + 9

$14^2 = 196 = 0196$

19

Division ?

- Lakukan pembagian pada key dengan operator modulus (sesuai dengan jumlah hash table).
- Jika key berupa string maka ubah menjadi number dengan cara :

U D I N

21 + 4 + 9 + 14

48 % 20

8

A D I

1 + 4 + 9

14 % 20

14

Folding ?

- Hash Function folding akan men-transformasi string ke hashed key dengan 2 Langkah :

1. Bagi key menjadi bagian-bagian yang memiliki jumlah digit sama.
2. Jumlahkan setiap bagian.

U	D	I	N
21	4	9	14

$$21 + 49 + 14$$

84

O	K	I
15	11	9

$$01 + 51 + 19$$

71

O	K	I
15	11	9

$$51 + 19$$

70



Folding ?

- Hash Function folding akan men-transformasi string ke hashed key dengan 2 Langkah :

1. Bagi key menjadi bagian-bagian yang memiliki jumlah digit sama.
2. Jumlahkan setiap bagian.

U	D	I	N
21	4	9	14

$$41 + 94 + 12$$

147

O	K	I
15	11	9

$$91 + 15$$

106

Digit Extraction ?

- Mendapatkan hashed key dengan cara mengambil digit-digit tertentu dari sebuah key .

U D I N
21 4 9 14

241

O K I
15 11 9

119

Rotating Hash ?

- Mendapatkan hashed key dengan cara membalik urutan dari key.

U D I N
21 4 9 14
4 1 9 4 2 1

O K I
15 11 9
9 1 1 5 1

Truncation ?

- Mendapatkan hashed key dengan cara memenggal key sebanyak k (tidak boleh dilewati) dari n digit, dimana $k < n$.

U	D	I	N
21	4	9	14
2 1 4			

U	D	I	N
21	4	9	14
1 4 9			

U	D	I	N
21	4	9	14
4 9 1 4			

Hash Function yang Baik ?

- Hash Function harus memiliki sifat berikut :
 - Mudah dihitung
 - Dua key yang berbeda akan dipetakan pada dua sel yang berbeda pada array. Tapi secara umum tidak bisa berlaku karena bisa jadi dua key yang berbeda mempunyai hasil hashed key yang sama (tabrakan).
 - Membagi key secara rata pada seluruh sel.
- Sebuah hash function sederhana adalah menggunakan fungsi modulus (sisa bagi) dengan bilangan prima.
- Dapat menggunakan manipulasi digit dengan kompleksitas rendah dan distribusi key yang rata.

Memilih Hash Function ?

- Sebuah hash function yang bagus memiliki dua kriteria :
 - Mudah dan cepat dihitung
 - Harus meminimalkan juga collisions (tabrakan) yang terjadi.

Collision ?

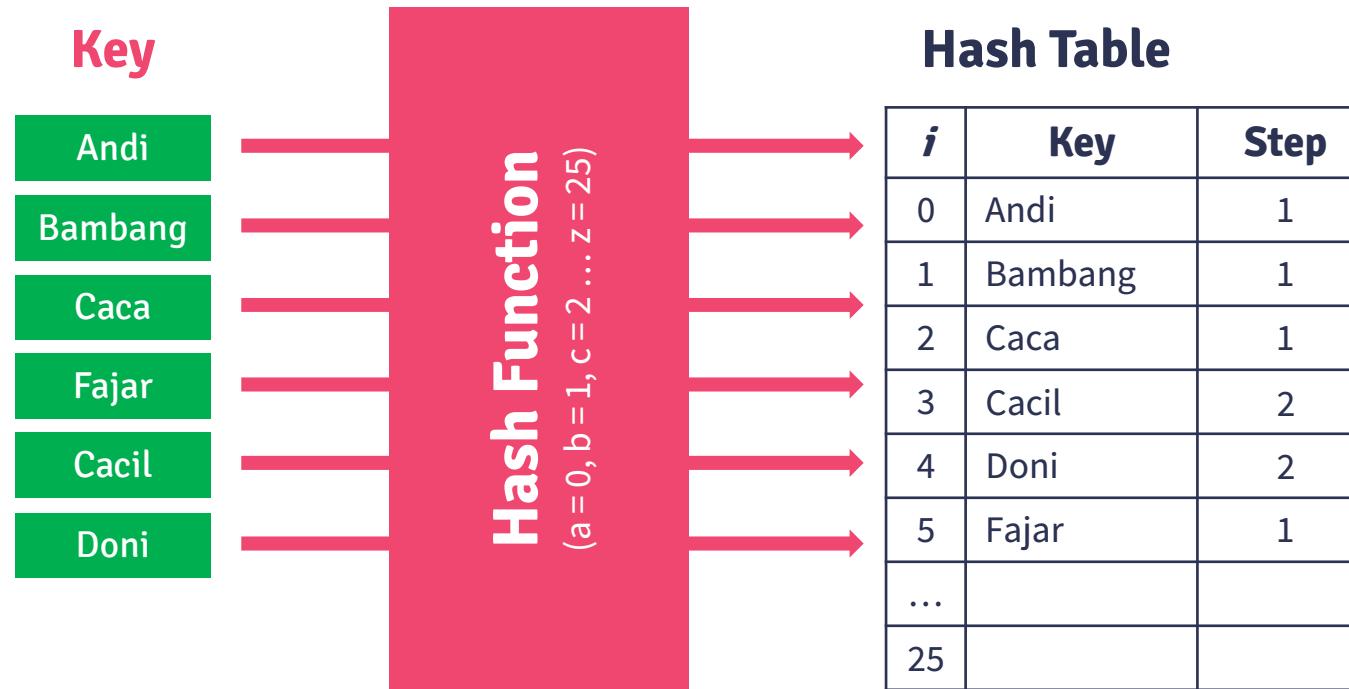
- Dikatakan terjadi collision (tabrakan) jika dua buah keys dipetakkan pada sebuah sel yang sama.
- Collision bisa terjadi saat melakukan insertion (penambahan data).
- Penyelesaian bila terjadi collision (tabrakan) disebut *Collision Resolution*.
- Dibutuhkan prosedur tambahan untuk mengatasi terjadinya collision (tabrakan).

Collision Resolution ?

- Ada dua strategi umum dalam melakukan Collision Resolution :
 - **Closed Hashing (Open Addressing)**
 - Linear Probing.
 - Quadratic Probing.
 - Double Hashing.
 - **Open Hashing (Chaining)**

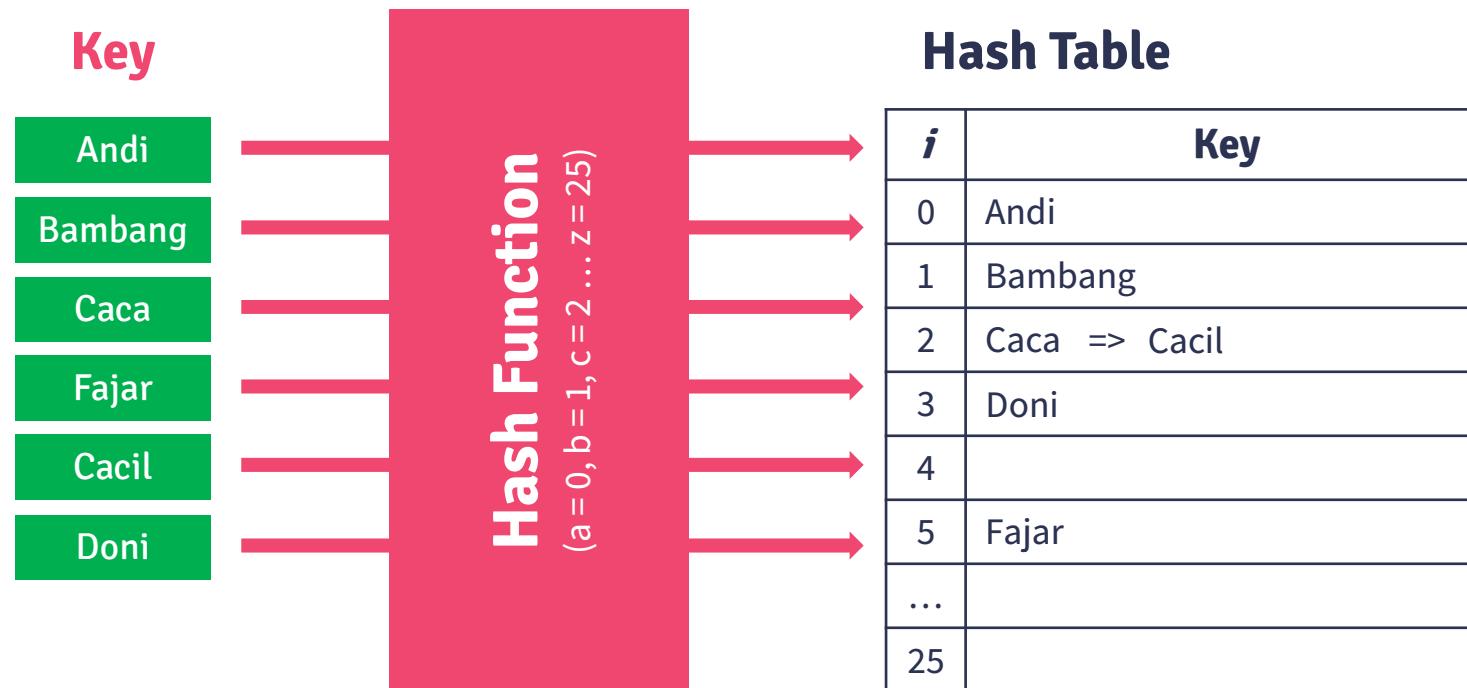
Closed Hashing – Linear Probing ?

- Ideanya adalah mencari alternatif sel lain pada hash table.



Open Hashing (Chaining) ?

- Ideanya meletakkan key pada table menggunakan linked list.





Video Selanjutnya

**Ngoding Program Login Sederhana
Dengan Hashing C++**



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 8

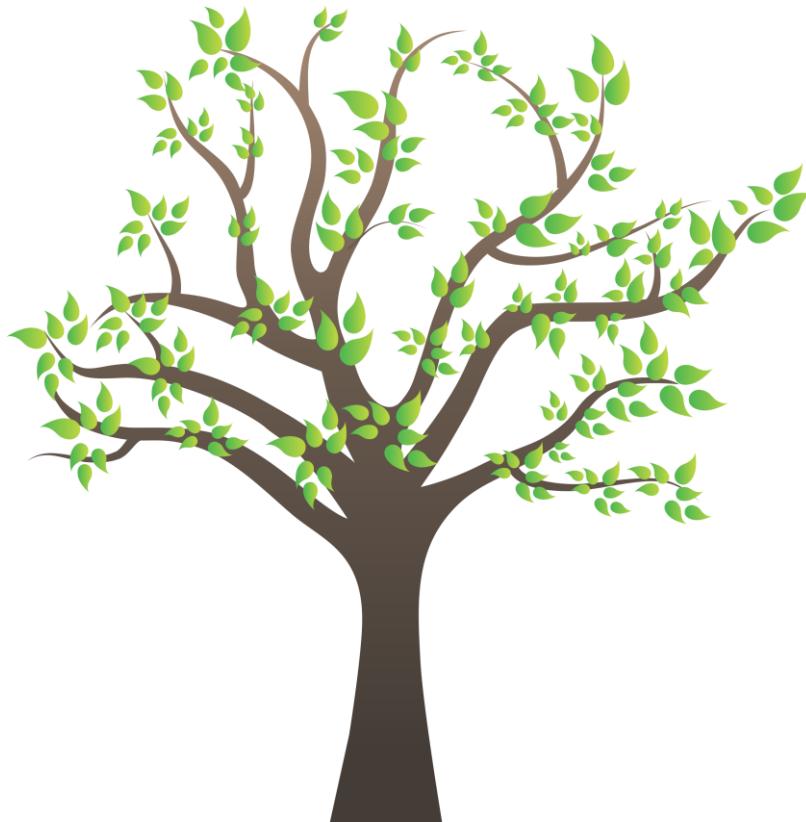
Tree



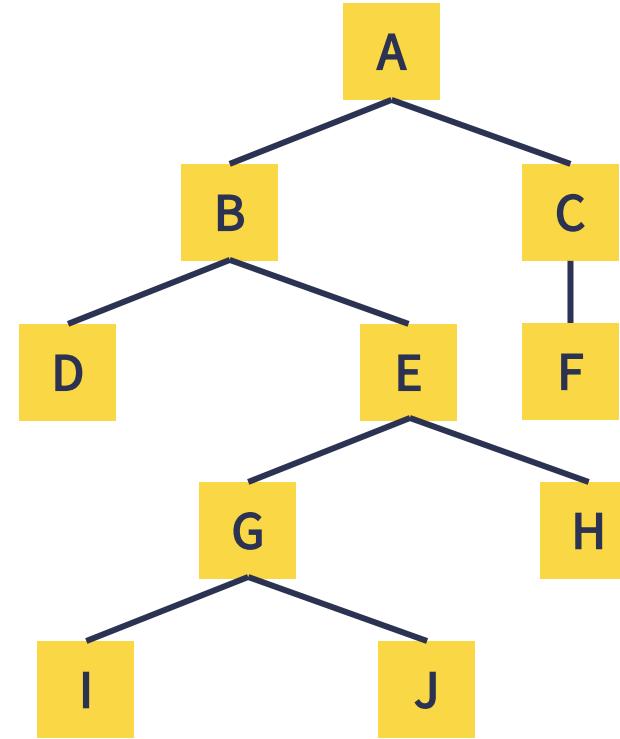
Yang dipelajari ?

- Konsep Tree
- Level & Derajat pada Tree
- Istilah dan Hubungan Komponen Tree
- Definisi Tree
- Ordered & Unordered Tree
- Konsep Binary Tree
- Jenis-jenis Binary Tree
- Tree Tranversal
- Operasi pada Tree

Sedikit Gambaran ?



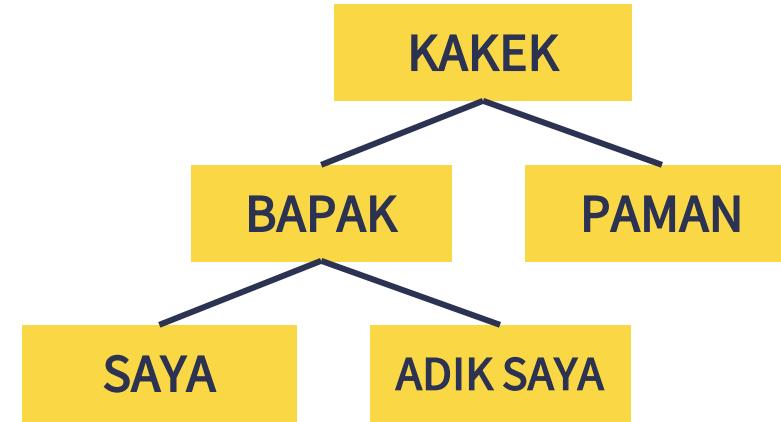
SEBENARNYA



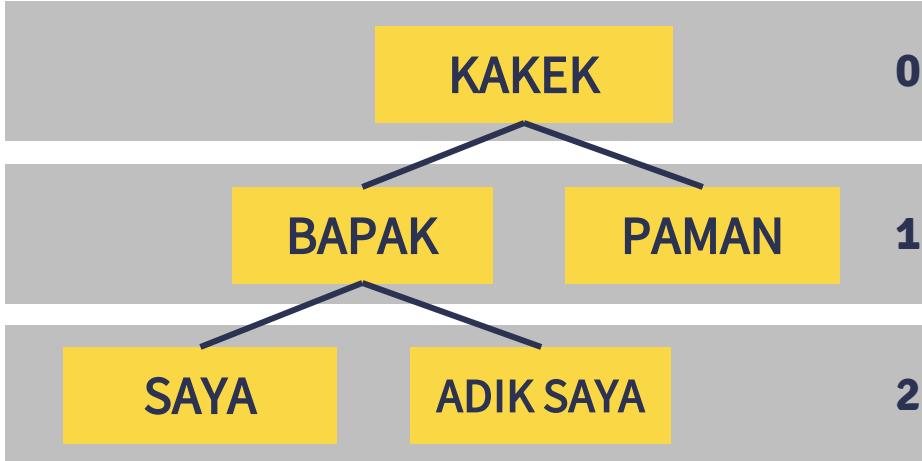
STRUKTUR DATA

Konsep Tree ?

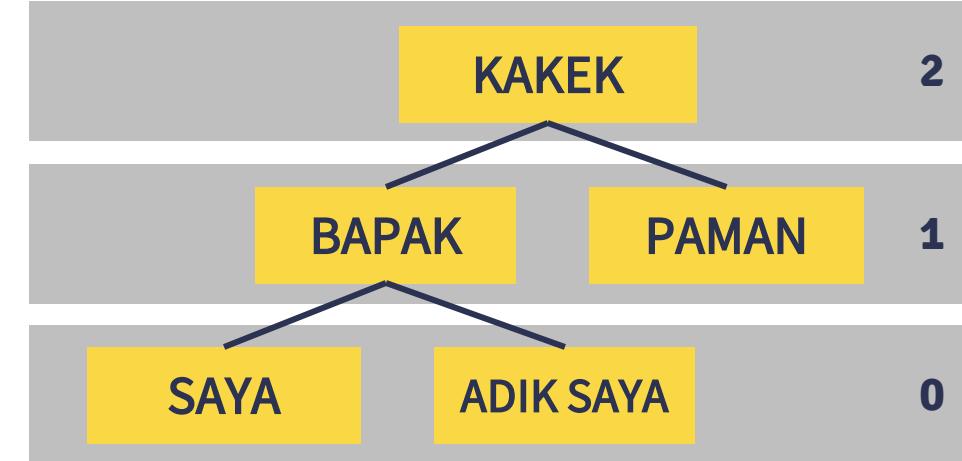
- Konsep struktur data yang terdiri dari akar dan simpul-simpul yang berada dibawahnya.
- Struktur data yang menunjukkan hubungan bertingkat (memiliki hirarki).
- Merupakan struktur data yang tidak linear yang digunakan untuk mempresentasikan data yang bersifat hirarki (urutan / tingkatan / kedudukan) antar elemen-elemennya.
- Contoh : struktur organisasi, silsilah keluarga, struktur folder, dll.



Level & Derajat Tree ?



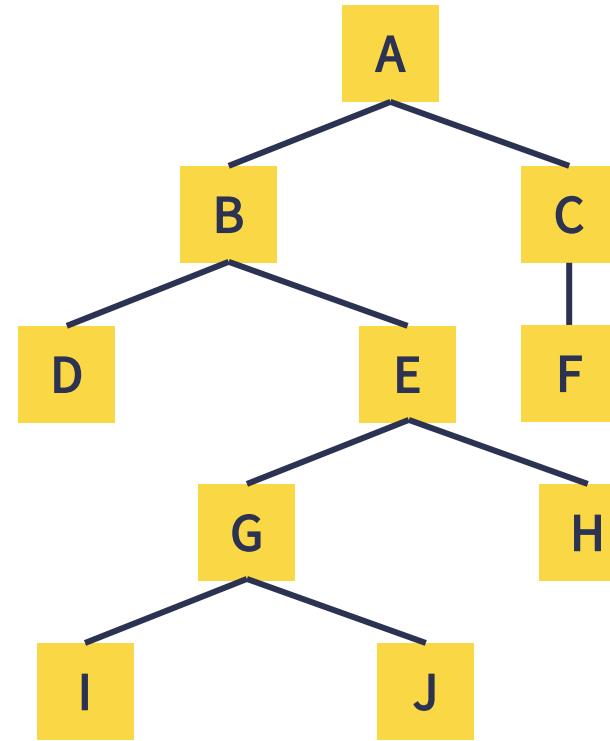
LEVEL



DERAJAT

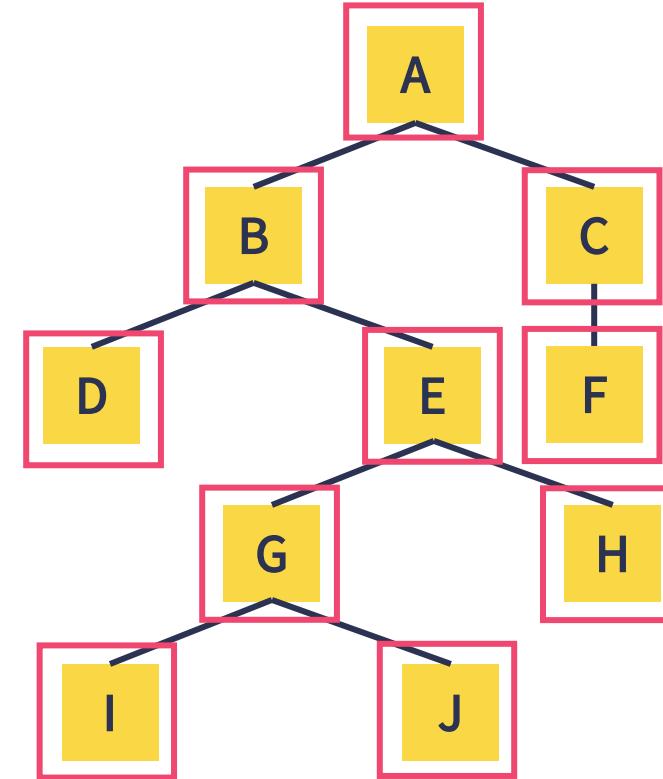
Istilah & Hubungan Komponen Tree ?

- **Node (Simpul)**
- **Predecessor (Pendahulu)**
- **Successor (Penerus)**
- **Ancestor (Leluhur)**
- **Descendant (Keturunan)**
- **Parent (Orang tua)**
- **Child (Anak)**
- **Sibling (Saudara)**
- **Subtree**
- **Size**
- **Height**
- **Root (Akar)**
- **Leaf (Daun)**
- **Degree**
- **Forest (Hutan)**
- **Depth (Kedalaman)**



Istilah & Hubungan Komponen Tree ?

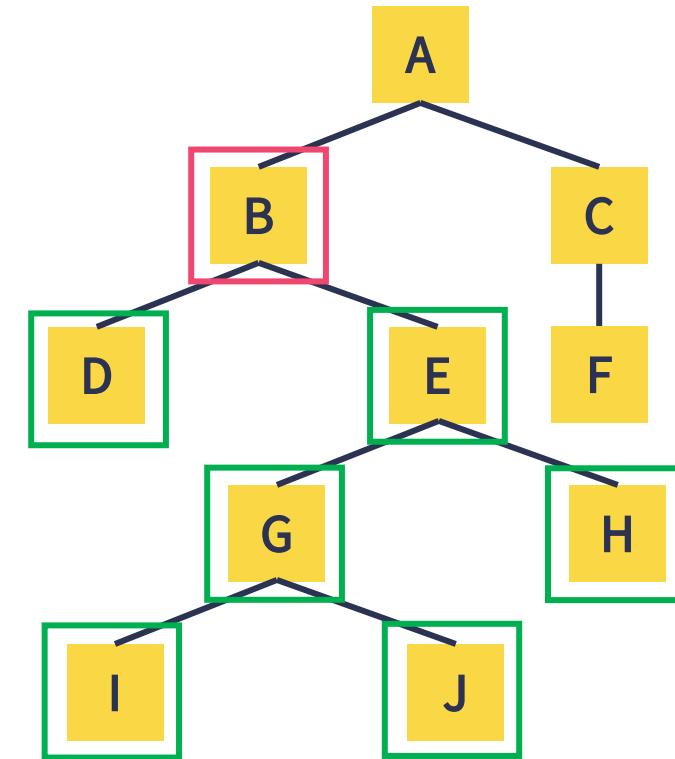
- **Node (Simpul)** : adalah simpul dari masing-masing data dari suatu tree.



Node = A,B,C,D,E,F,G,H,I,J

Istilah & Hubungan Komponen Tree ?

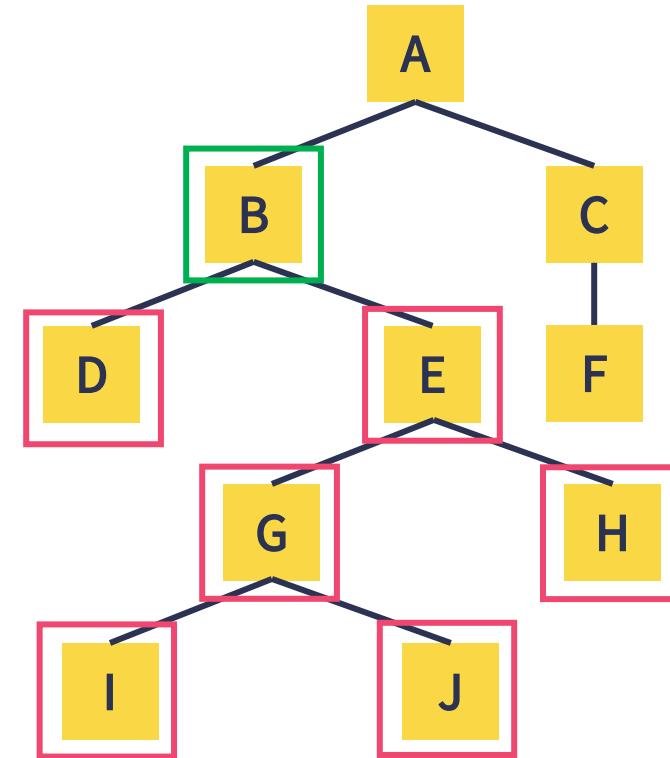
- **Predecessor** (Pendahulu) : adalah node yang berada diatas node tertentu



Predecessor (D,E,G,H,I,J) = B

Istilah & Hubungan Komponen Tree ?

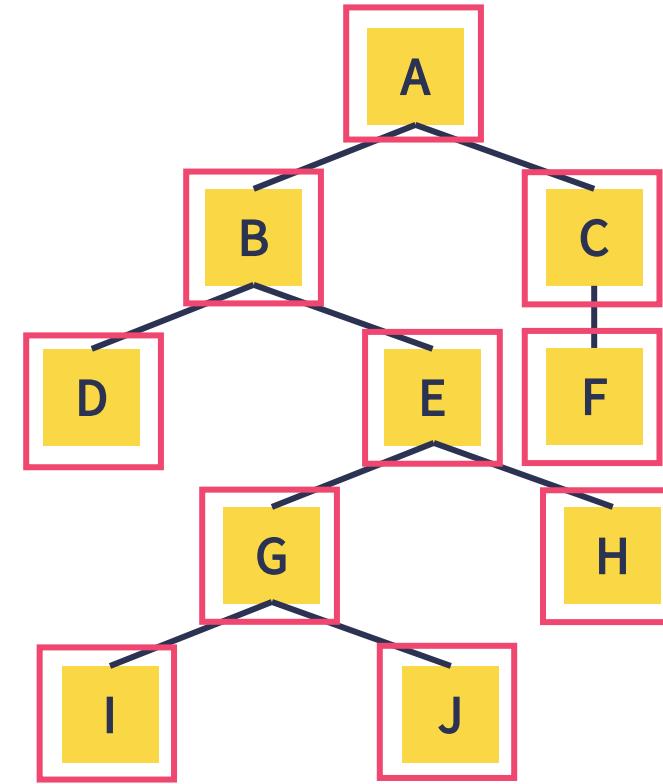
- **Successor (Penerus) & Subtree** : adalah node yang berada dibawah node tertentu



Successor (B) = D,E,G,H,I,J

Istilah & Hubungan Komponen Tree ?

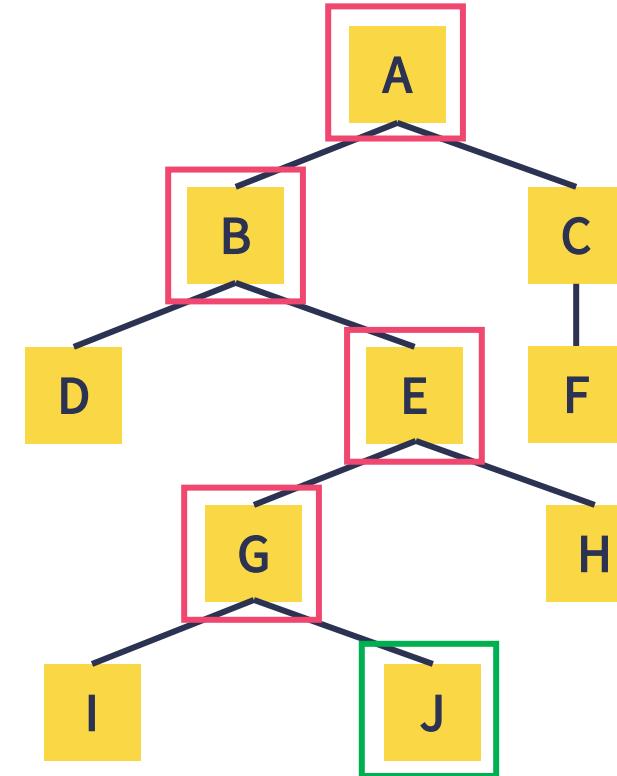
- **Size** : adalah banyaknya node disebuah tree



Size = 10

Istilah & Hubungan Komponen Tree ?

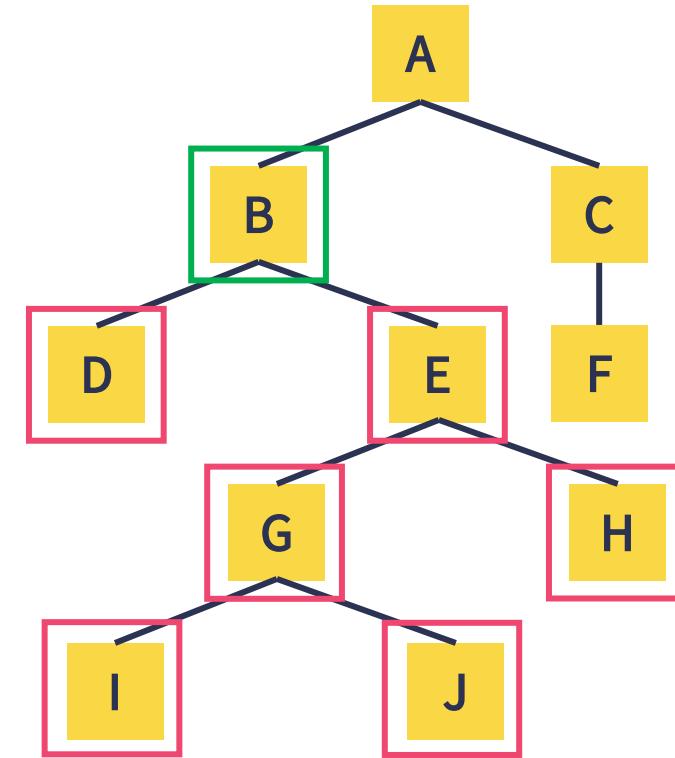
- **Ancestor (Leluhur)** : Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.



Ancestor (J) = G,E,B,A

Istilah & Hubungan Komponen Tree ?

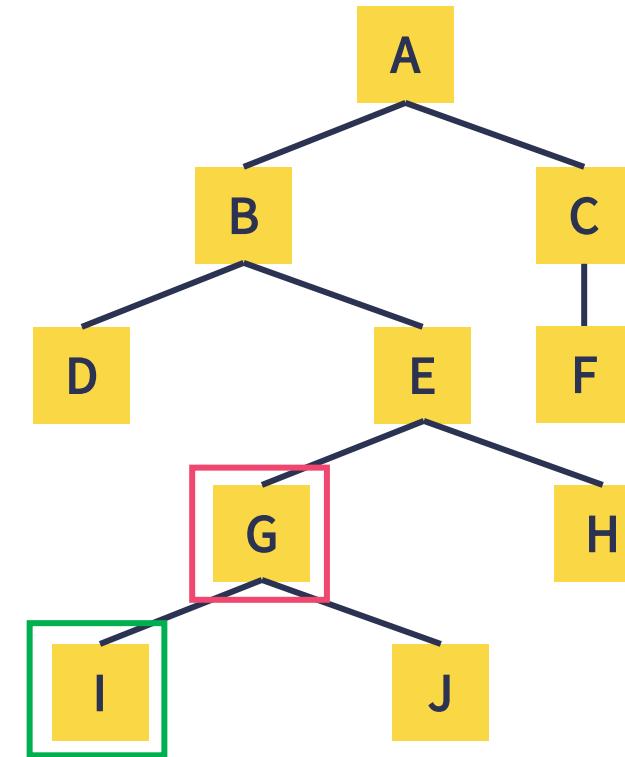
- **Descendant (Keturunan)** : Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama.



Descendant (B) = D,E,G,H,I,J

Istilah & Hubungan Komponen Tree ?

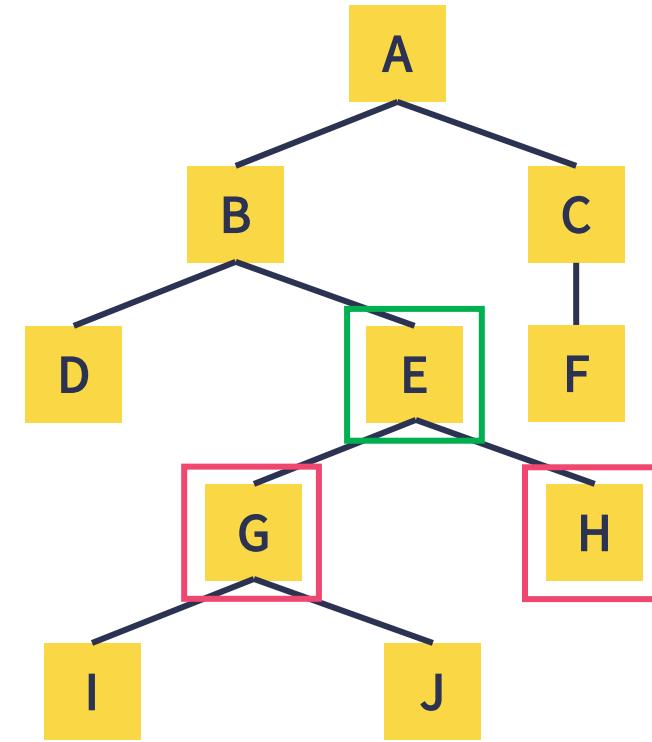
- **Parent (Orang tua)** : Predecessor (pendahulu) satu level diatas suatu node.



Parent (I) = G

Istilah & Hubungan Komponen Tree ?

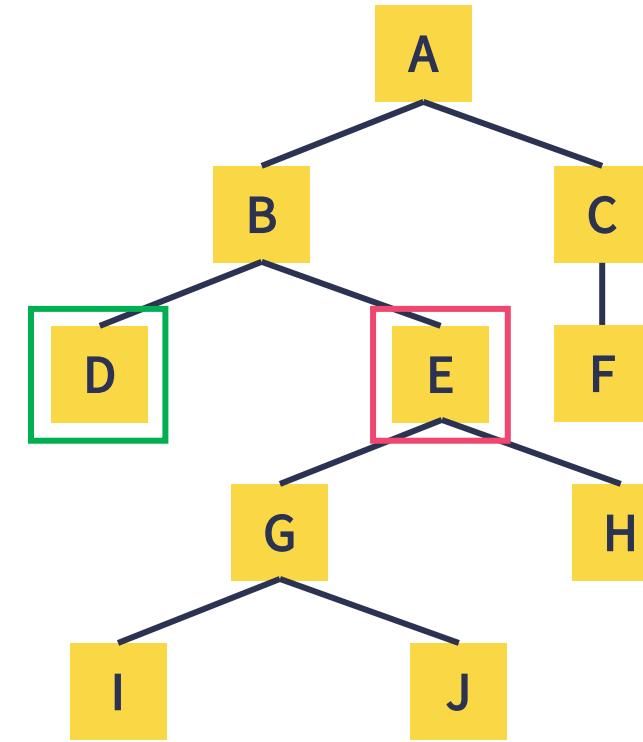
- **Child (Anak)** : adalah successor (penerus) satu level dibawah suatu node.



Child (E) = G,H

Istilah & Hubungan Komponen Tree ?

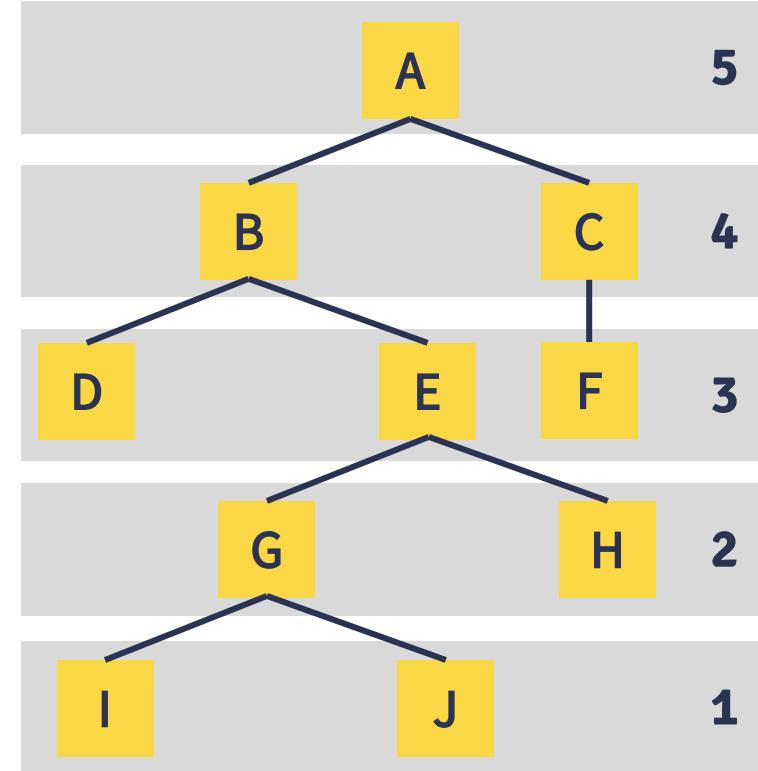
- **Sibling (Saudara)** : adalah node-node yang memiliki parent yang sama.



Sibling (D) = E

Istilah & Hubungan Komponen Tree ?

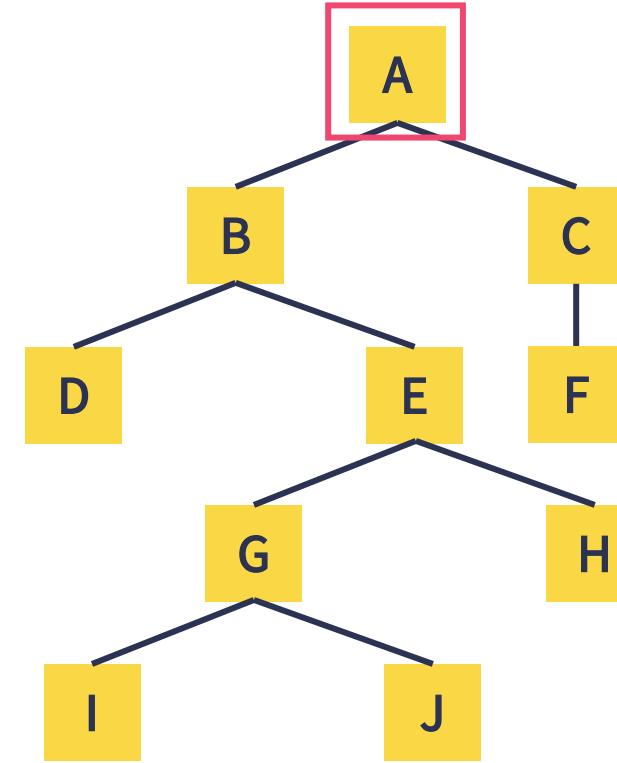
- **Height** : Banyaknya tingkatan dalam suatu tree.



Height = 5

Istilah & Hubungan Komponen Tree ?

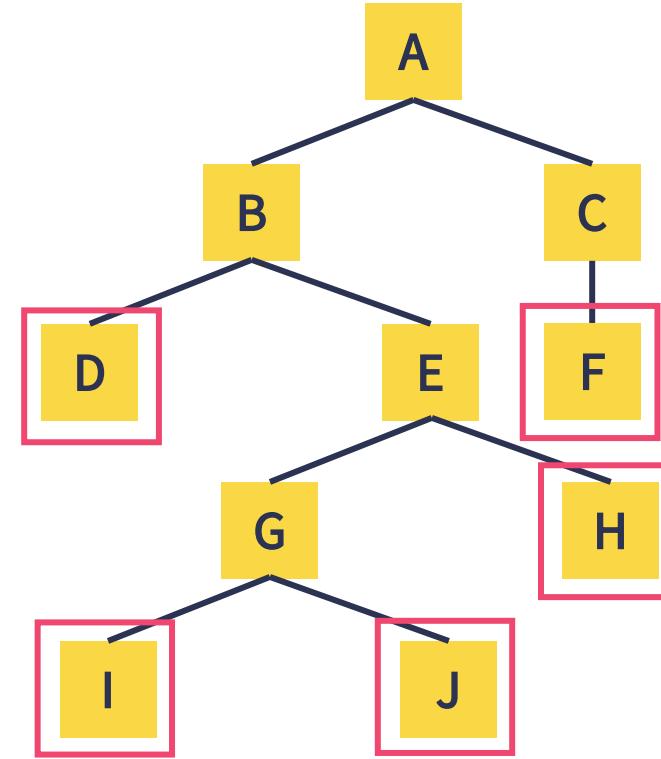
- **Root (Akar)** : adalah node khusus yang tidak memiliki predecessor (pendahulu).



Root = A

Istilah & Hubungan Komponen Tree ?

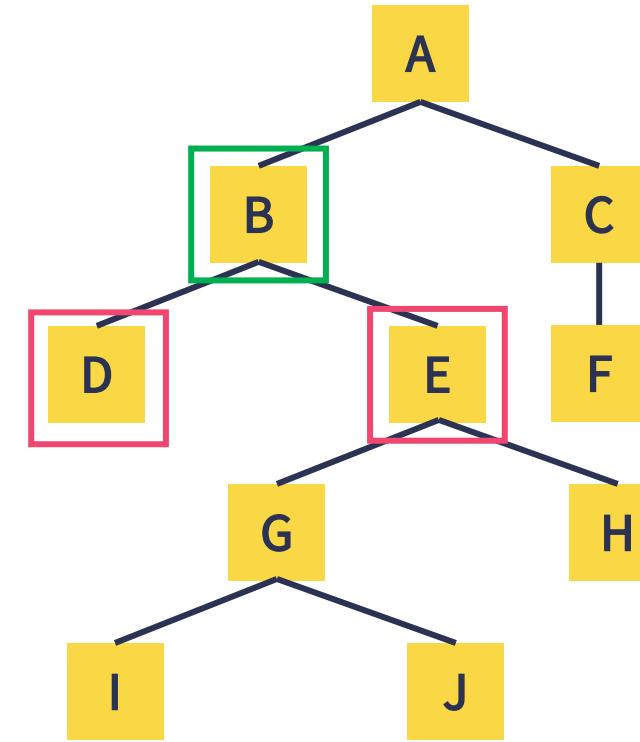
- **Leaf (Daun)** : adalah node-node dalam tree yang tidak memiliki successor (penerus).



Leaf = D,F,H,I,J

Istilah & Hubungan Komponen Tree ?

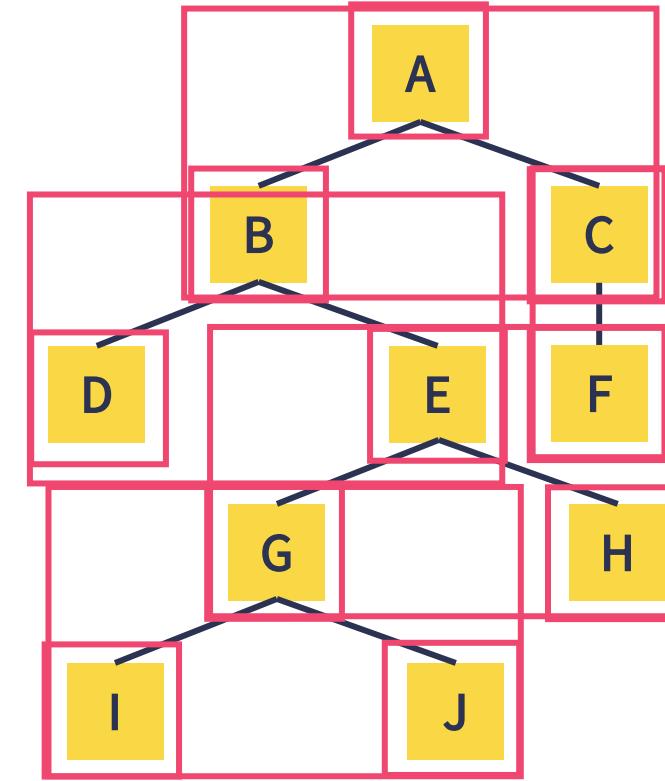
- **Degree** : adalah banyaknya child (anak) dalam suatu node.



Degree (B) = 2

Istilah & Hubungan Komponen Tree ?

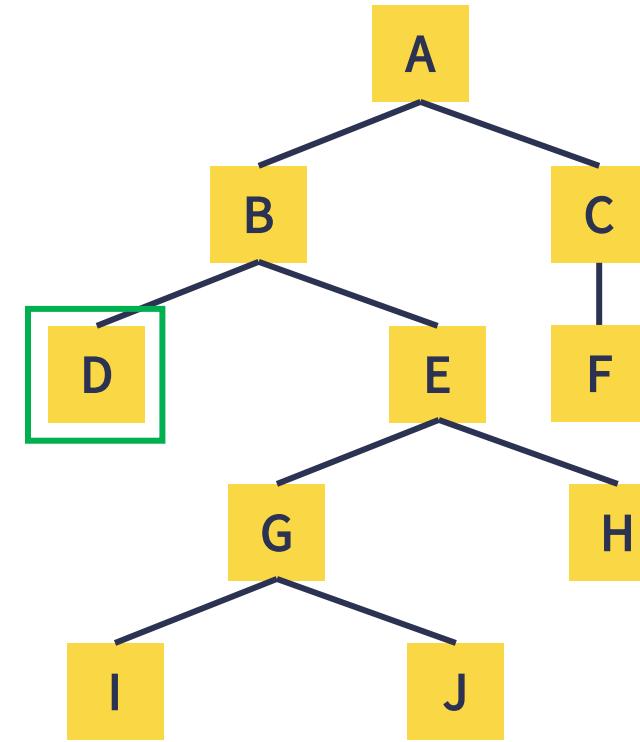
- **Forest (Hutan)** : adalah kumpulan dari tree.



Forest = 15

Istilah & Hubungan Komponen Tree ?

- **Depth (Kedalaman)** : adalah hasil tingkat node maksimum dikurang satu (level dari node x).



Depth (D) = 2

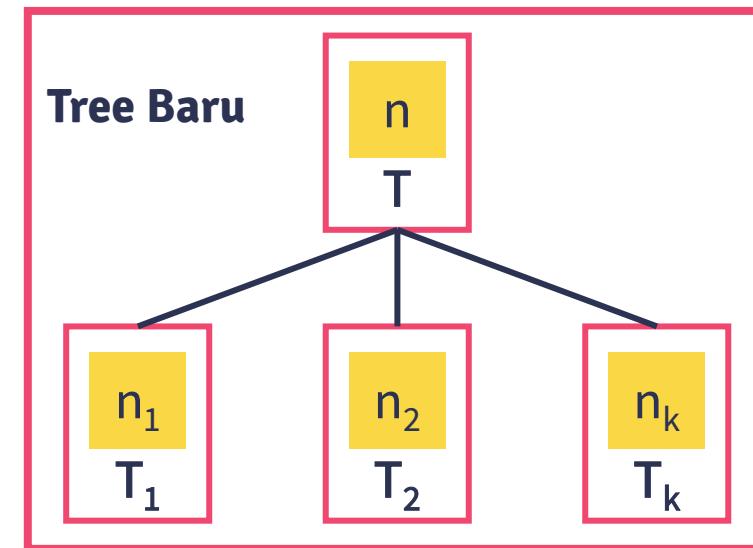
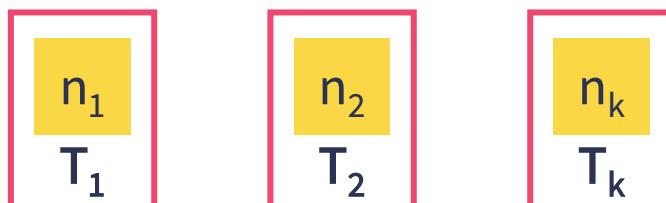
Definisi Tree ?

- Sebuah tree didefinisikan sebagai struktur yang dibentuk secara rekursif oleh aturan berikut.
 - Sebuah node adalah sebuah tree. Node satu-satunya pada tree ini berfungsi sebagai root maupun leaf.
 - Dari k buah tree $T_1 \sim T_k$, dan masing-masing memiliki root $n_1 \sim n_k$.
 - Jika node n adalah parent dari $n_1 \sim n_k$, akan diperoleh sebuah tree baru T yang memiliki root n. Dalam kondisi ini, tree $T_1 \sim T_k$ menjadi subtree dari tree T.

T : Tree

N : Node

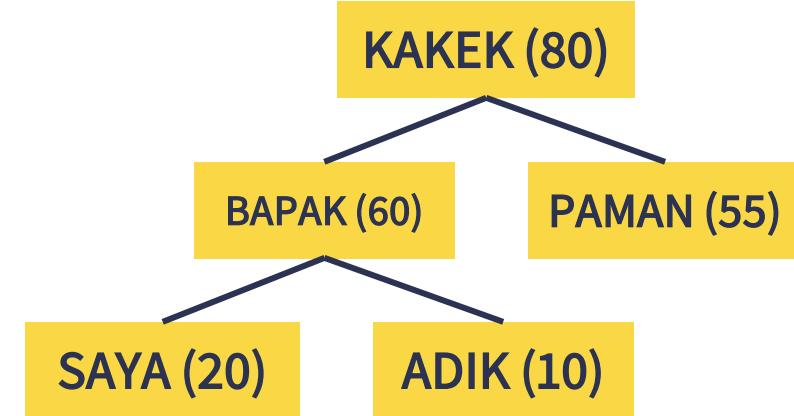
k : banyak



Ordered & Unordered Tree ?

- Ordered Tree

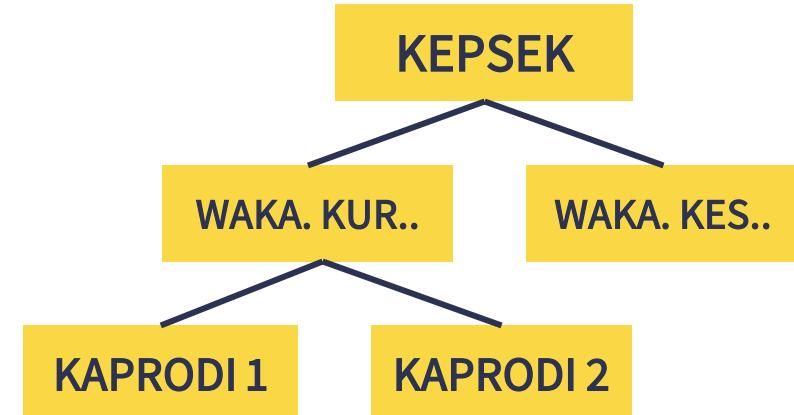
- Antar sibling (saudara) terdapat urutan “usia”.
- Node yang paling kiri berusia paling tua (sulung), sedangkan node yang paling kanan berusia paling muda (bungsu).
- Posisi node diatur atas urutan tertentu.
- Contoh : silsilah keluarga.



Ordered Tree

- Unordered Tree

- Antar sibling (saudara) tidak terdapat urutan tertentu.
- Contoh : struktur organisasi sekolah SMK.

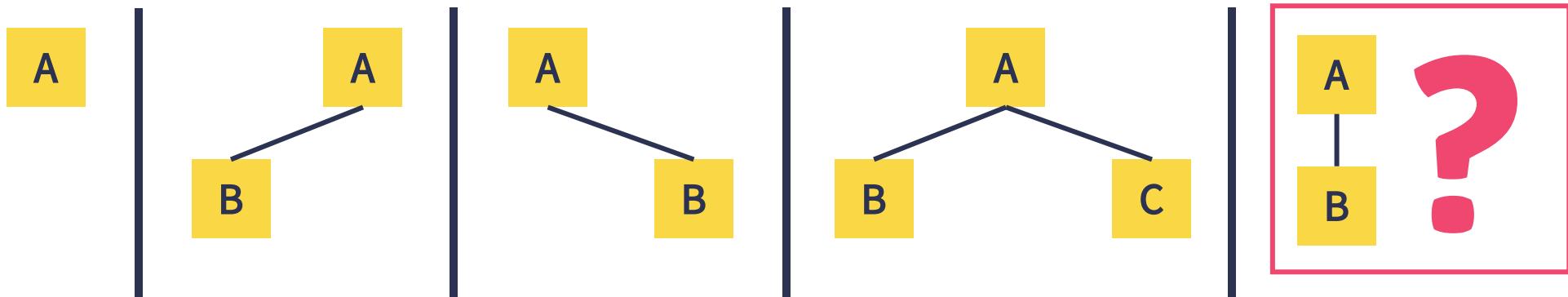


Unordered Tree

Konsep Binary Tree ?

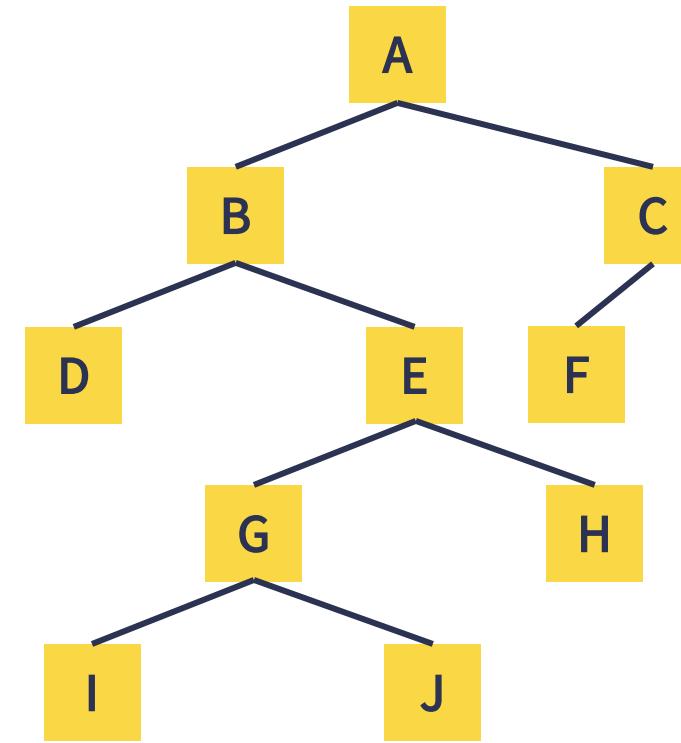
- Binary adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree harus terpisah.
 - Binary tree boleh tidak memiliki child (anak) ataupun subtree.
 - Boleh hanya memiliki subtree sebalah kiri (left subtree).
 - Boleh hanya memiliki subtree sebalah kanan (right subtree).
 - Boleh hanya memiliki subtree sebalah kiri (left subtree) dan kanan (right subtree).

Hati-hati menggambarkan Binary Tree



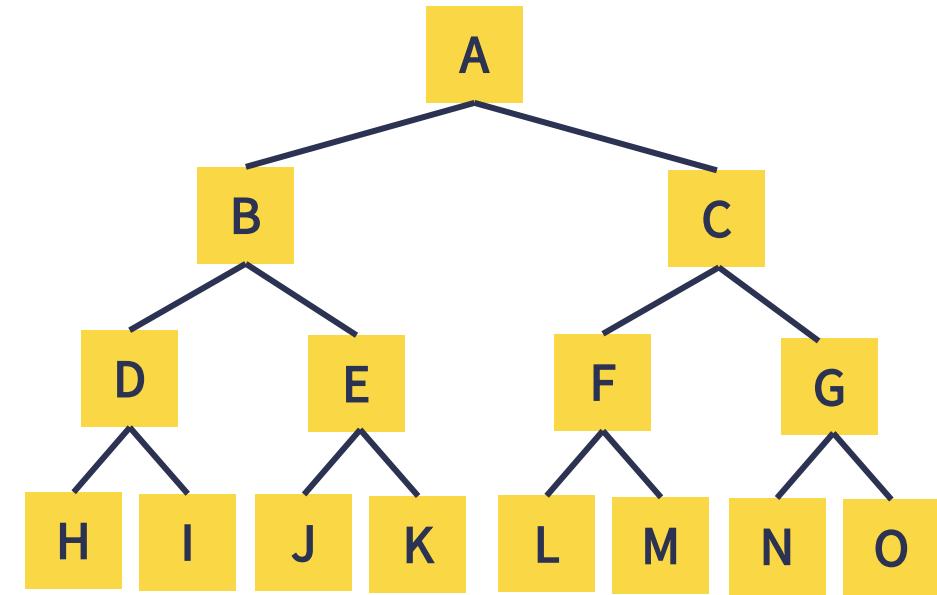
Jenis-jenis Binary Tree ?

- Full Binary Tree
- Complete Binary Tree
- Skewed Binary Tree



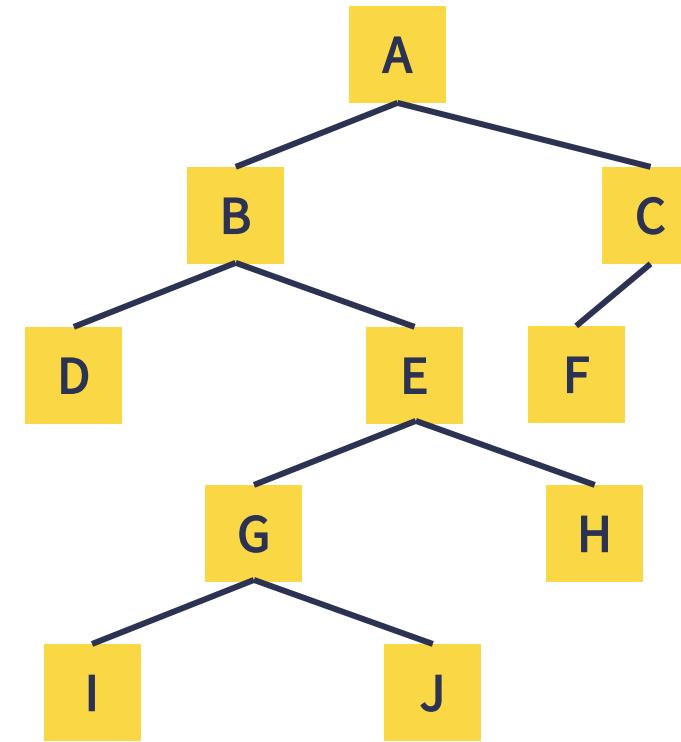
Jenis-jenis Binary Tree ?

- **Full Binary Tree :** adalah binary tree yang tiap node nya (kecuali leaf) memiliki dua child dan tiap subtree mempunyai panjang patch yang sama.



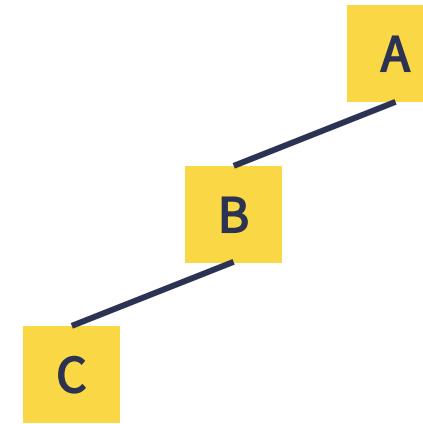
Jenis-jenis Binary Tree ?

- **Complete Binary Tree :** adalah binary tree yang mirip dengan full binary tree, namun setiap subtree boleh memiliki panjang patch yang berbeda.



Jenis-jenis Binary Tree ?

- **Skewed Binary Tree** : adalah binary tree yang semua node nya (kecuali left) hanya memiliki satu child.



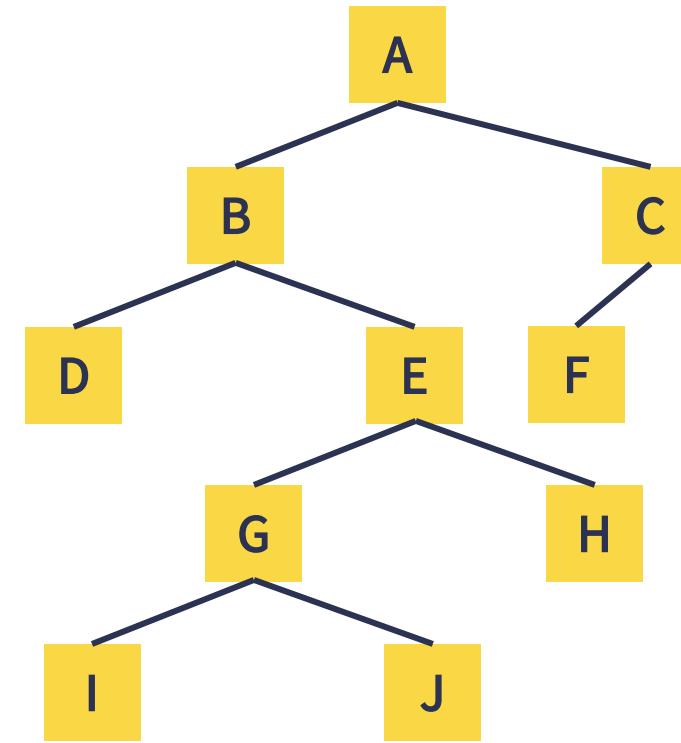
Definisi Tree Tranversal ?

- Adalah teknik menyusuri tiap node dalam sebuah tree secara sistematis, sehingga semua node dapat dan hanya satu kali saja dikunjungi.
- Ada tiga cara tranversal :
 - preOrder.
 - inOrder.
 - postOrder.
- Untuk tree atau node yang kosong, tranversal tidak perlu dilakukan.

Cara Tranversal ?

preOrder

1. Kunjungi root nya.
2. Telusuri subtree kiri.
3. Telusuri subtree kanan.

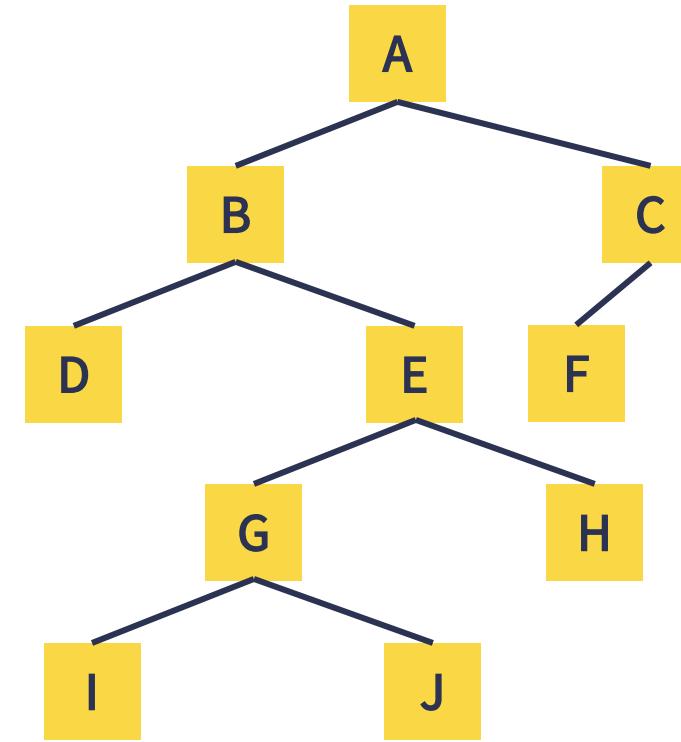


preOrder = A,B,D,E,G,I,J,H,C,F

Cara Tranversal ?

inOrder

1. Telusuri subtree kiri.
2. Kunjungi root nya.
3. Telusuri subtree kanan.

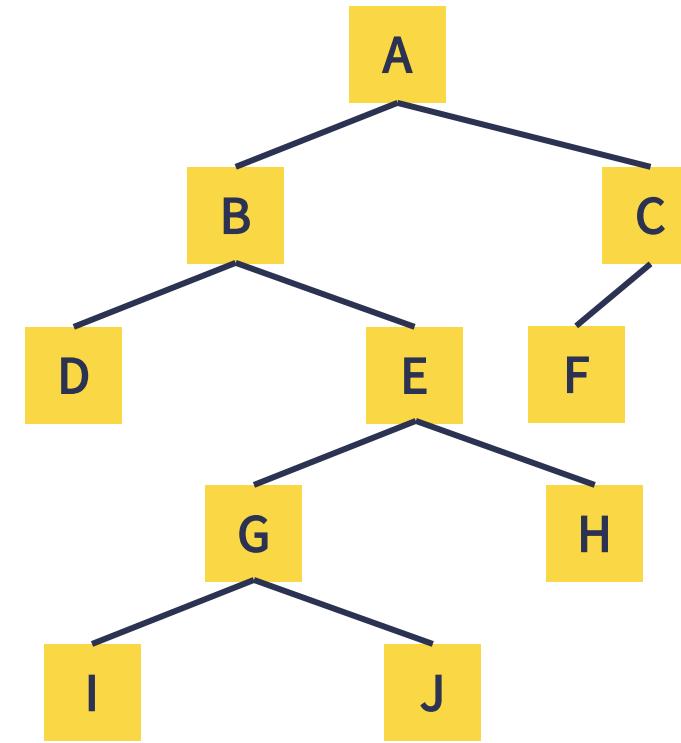


inOrder = D,B,I,G,J,E,H,A,F,C

Cara Tranversal ?

postOrder

1. Telusuri subtree kiri.
2. Telusuri subtree kanan.
3. Kunjungi root nya.



postOrder = **D,I,J,G,H,E,B,F,C,A**

Operasi pada Tree ?

- **Create** : digunakan untuk membentuk binary tree baru yang masih kosong.
- **Clear** : digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- **Empty** : digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- **Insert** : digunakan untuk memasukkan sebuah node kedalam tree.
- **Find** : digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- **Update** : digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- **Retrieve** : digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Delete Sub** : digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Characteristic** : digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average length-nya.
- **Traverse** : digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal.



Video Selanjutnya

Implementasi Tree C++



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 8

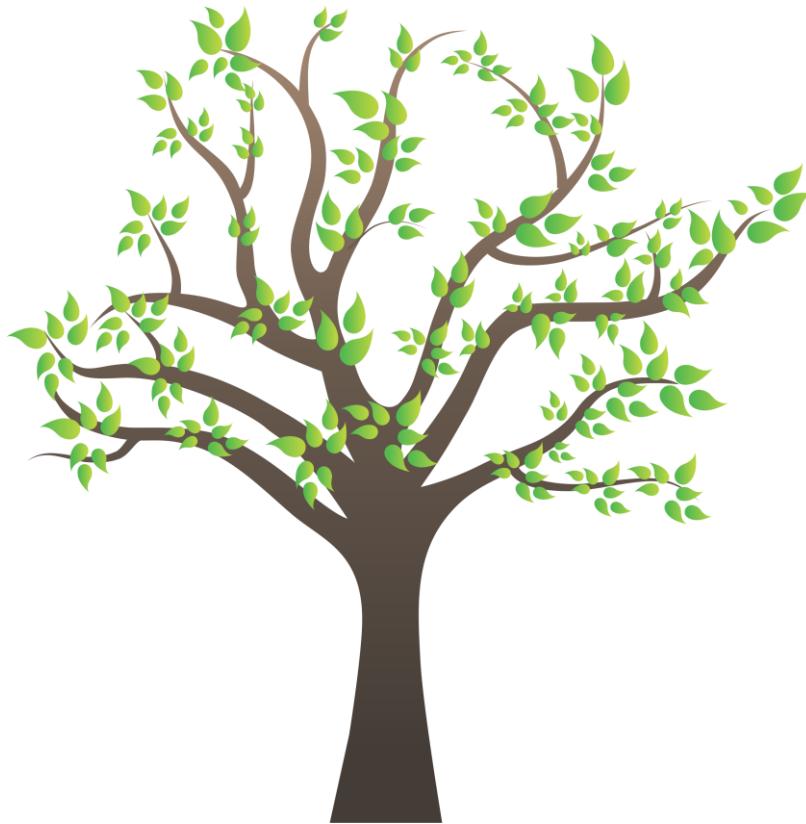
Tree



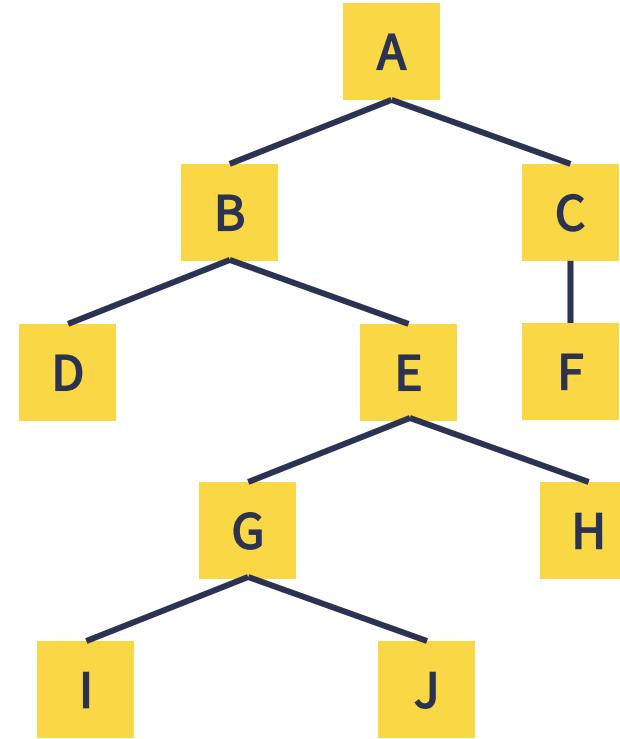
Yang dipelajari ?

- Konsep Tree
- Level & Derajat pada Tree
- Istilah dan Hubungan Komponen Tree
- Definisi Tree
- Ordered & Unordered Tree
- Konsep Binary Tree
- Jenis-jenis Binary Tree
- Tree Tranversal
- Operasi pada Tree

Sedikit Gambaran ?



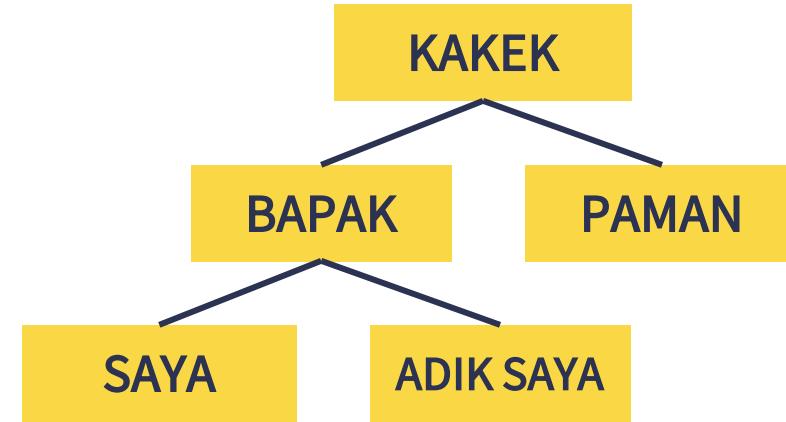
SEBENARNYA



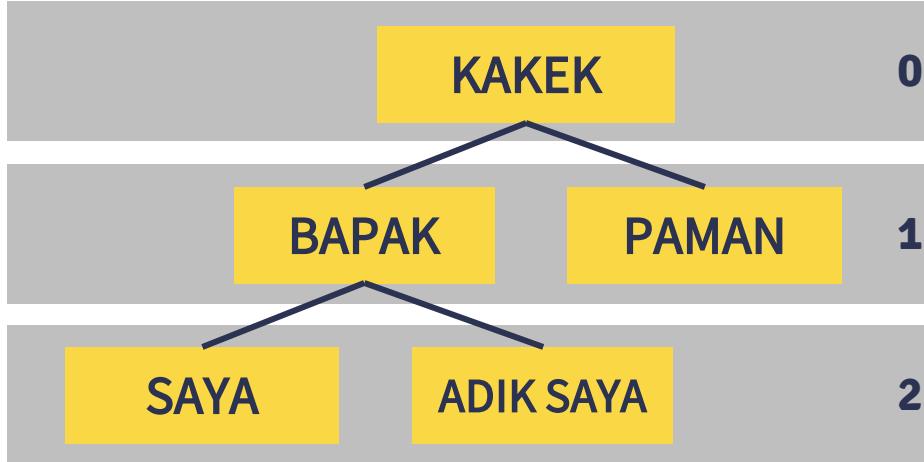
STRUKTUR DATA

Konsep Tree ?

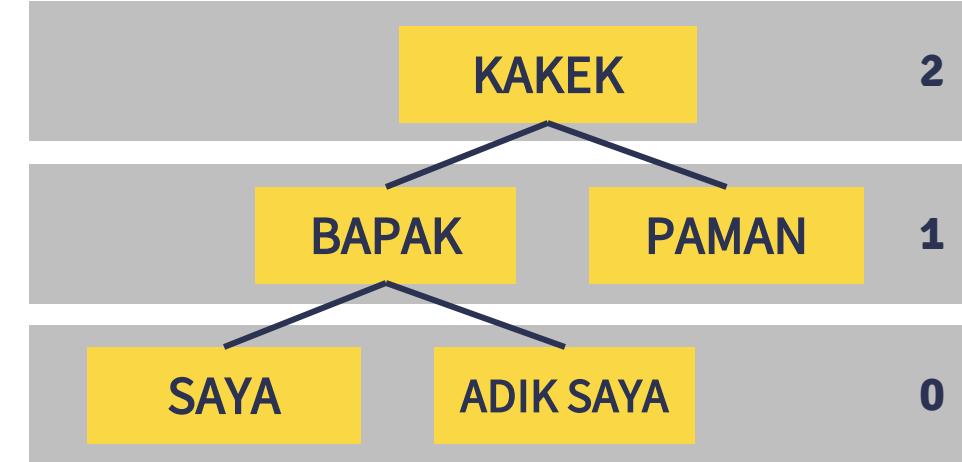
- Konsep struktur data yang terdiri dari akar dan simpul-simpul yang berada dibawahnya.
- Struktur data yang menunjukkan hubungan bertingkat (memiliki hirarki).
- Merupakan struktur data yang tidak linear yang digunakan untuk mempresentasikan data yang bersifat hirarki (urutan / tingkatan / kedudukan) antar elemen-elemennya.
- Contoh : struktur organisasi, silsilah keluarga, struktur folder, dll.



Level & Derajat Tree ?



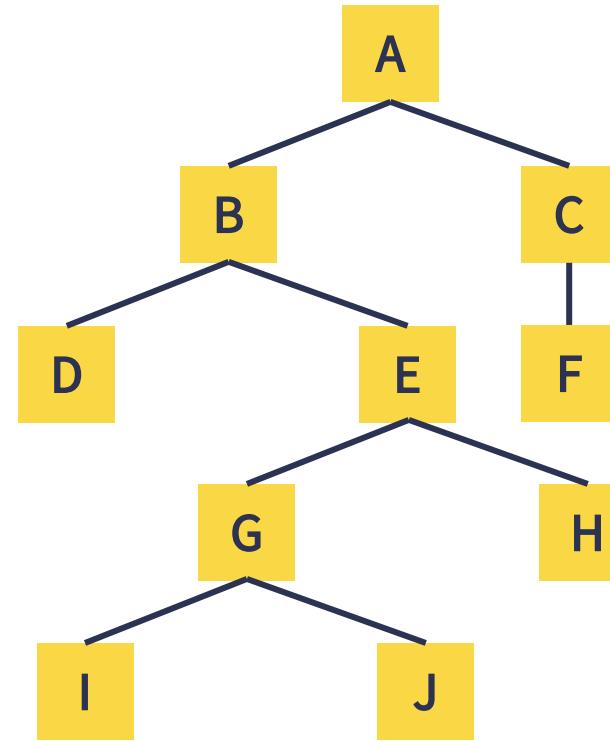
LEVEL



DERAJAT

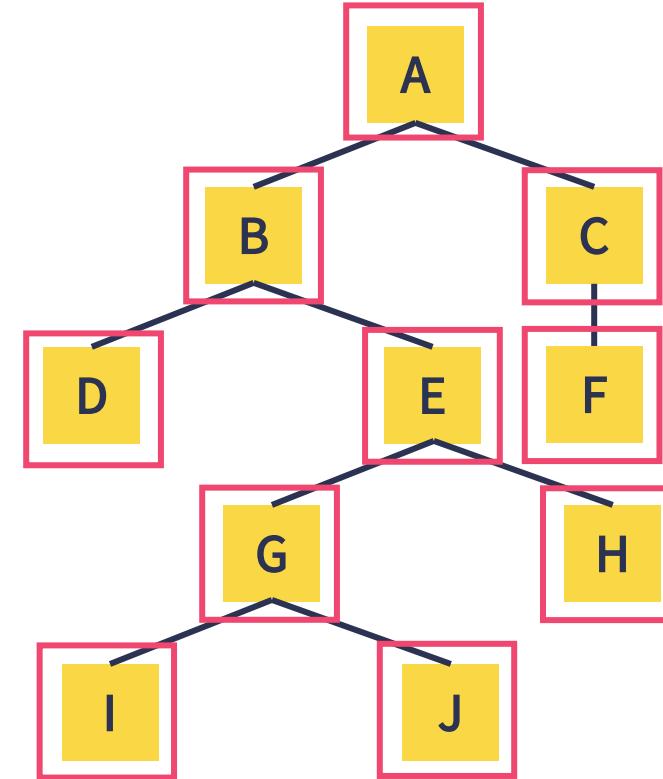
Istilah & Hubungan Komponen Tree ?

- **Node (Simpul)**
- **Predecessor (Pendahulu)**
- **Successor (Penerus)**
- **Ancestor (Leluhur)**
- **Descendant (Keturunan)**
- **Parent (Orang tua)**
- **Child (Anak)**
- **Sibling (Saudara)**
- **Subtree**
- **Size**
- **Height**
- **Root (Akar)**
- **Leaf (Daun)**
- **Degree**
- **Forest (Hutan)**
- **Depth (Kedalaman)**



Istilah & Hubungan Komponen Tree ?

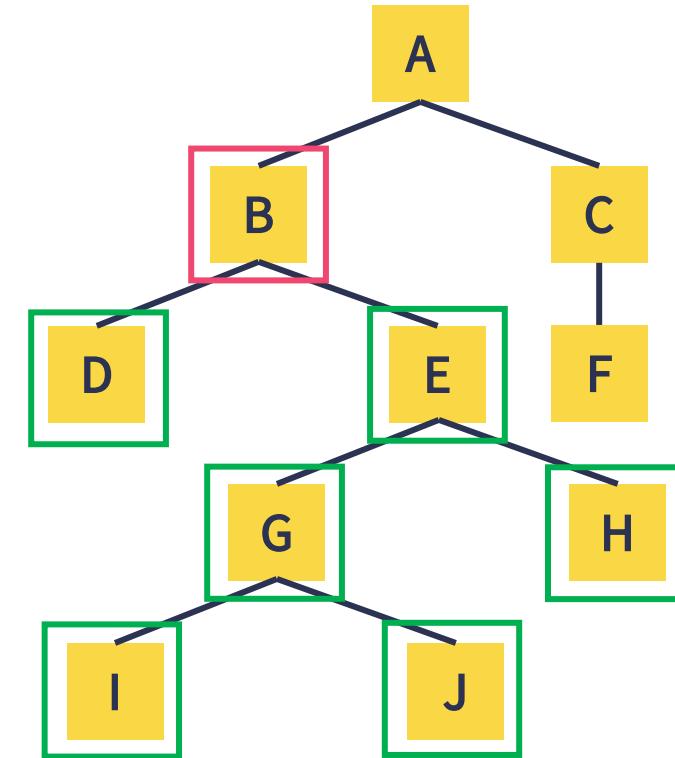
- **Node (Simpul)** : adalah simpul dari masing-masing data dari suatu tree.



Node = A,B,C,D,E,F,G,H,I,J

Istilah & Hubungan Komponen Tree ?

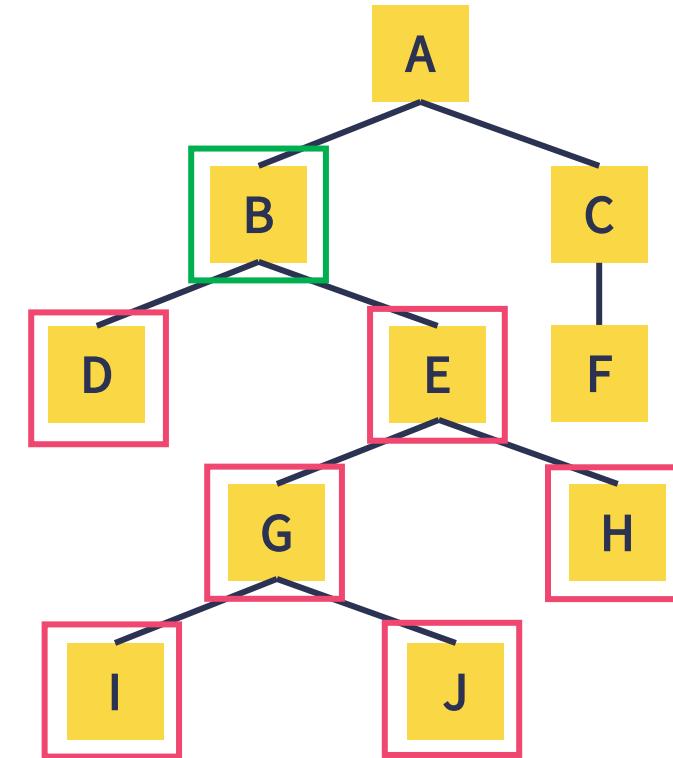
- **Predecessor** (Pendahulu) : adalah node yang berada diatas node tertentu



Predecessor (D,E,G,H,I,J) = B

Istilah & Hubungan Komponen Tree ?

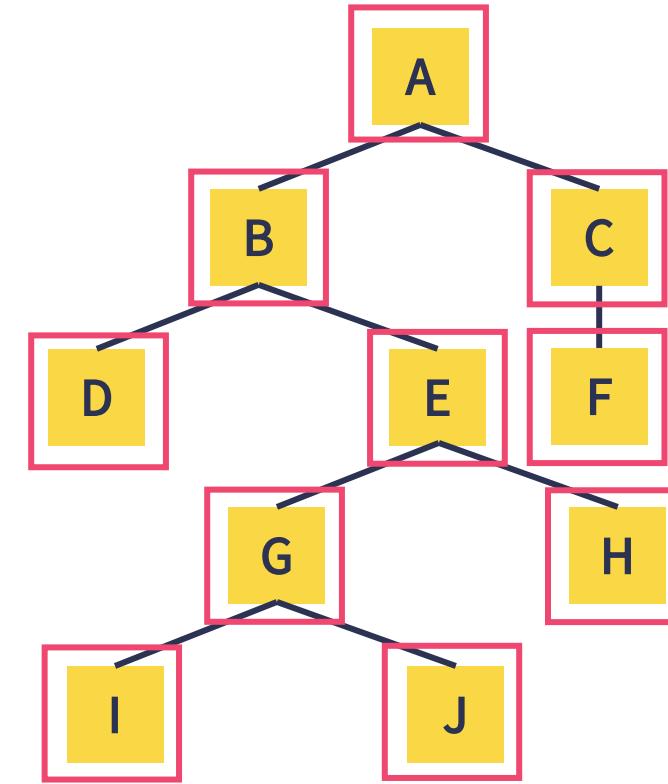
- **Successor (Penerus) & Subtree** : adalah node yang berada dibawah node tertentu



Successor (B) = D,E,G,H,I,J

Istilah & Hubungan Komponen Tree ?

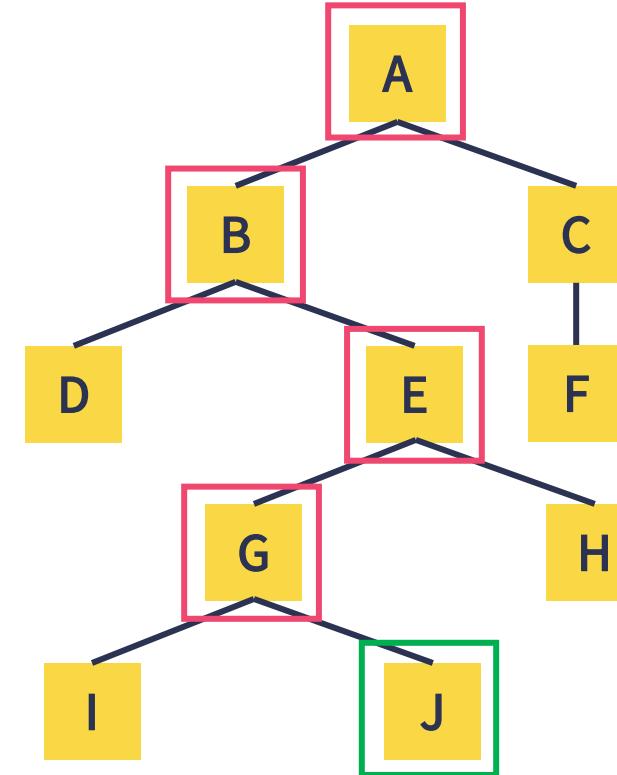
- **Size** : adalah banyaknya node disebuah tree



Size = 10

Istilah & Hubungan Komponen Tree ?

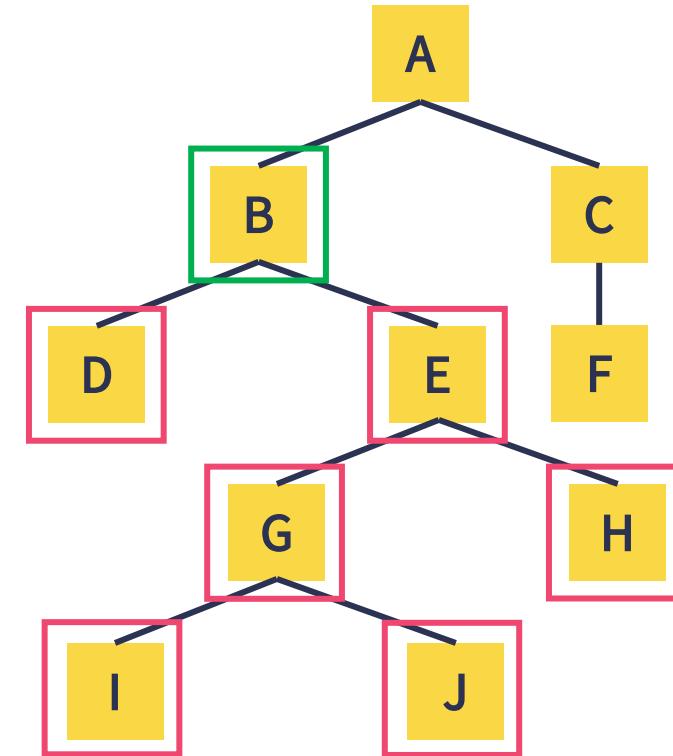
- **Ancestor (Leluhur)** : Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.



Ancestor (J) = G,E,B,A

Istilah & Hubungan Komponen Tree ?

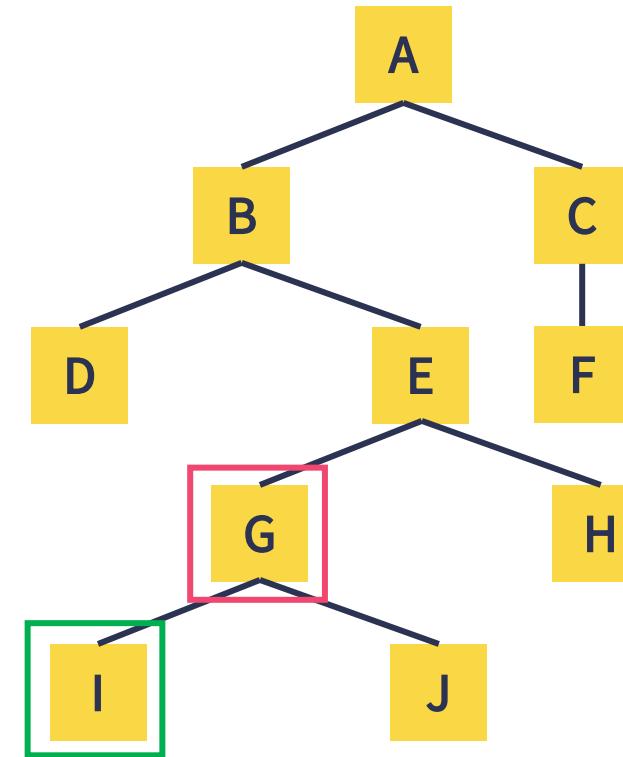
- **Descendant (Keturunan)** : Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama.



Descendant (B) = D,E,G,H,I,J

Istilah & Hubungan Komponen Tree ?

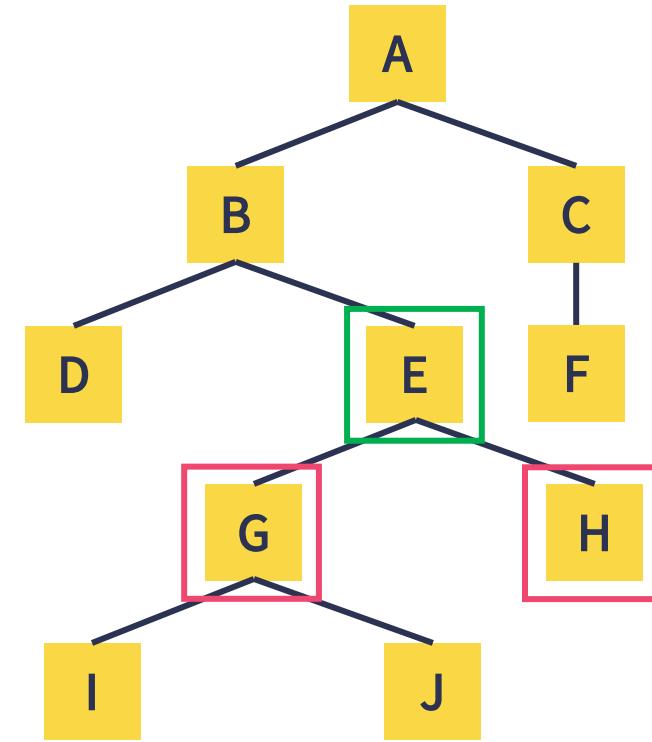
- **Parent (Orang tua)** : Predecessor (pendahulu) satu level diatas suatu node.



Parent (I) = G

Istilah & Hubungan Komponen Tree ?

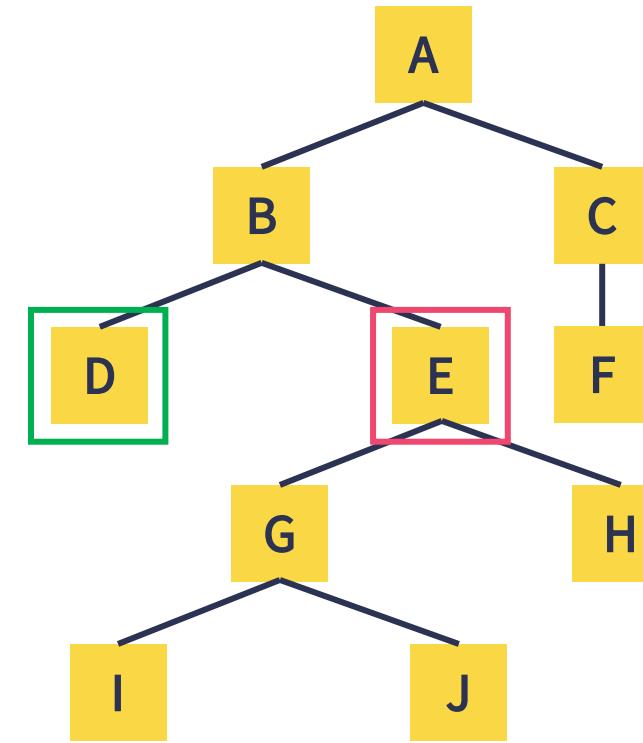
- **Child (Anak)** : adalah successor (penerus) satu level dibawah suatu node.



Child (E) = G,H

Istilah & Hubungan Komponen Tree ?

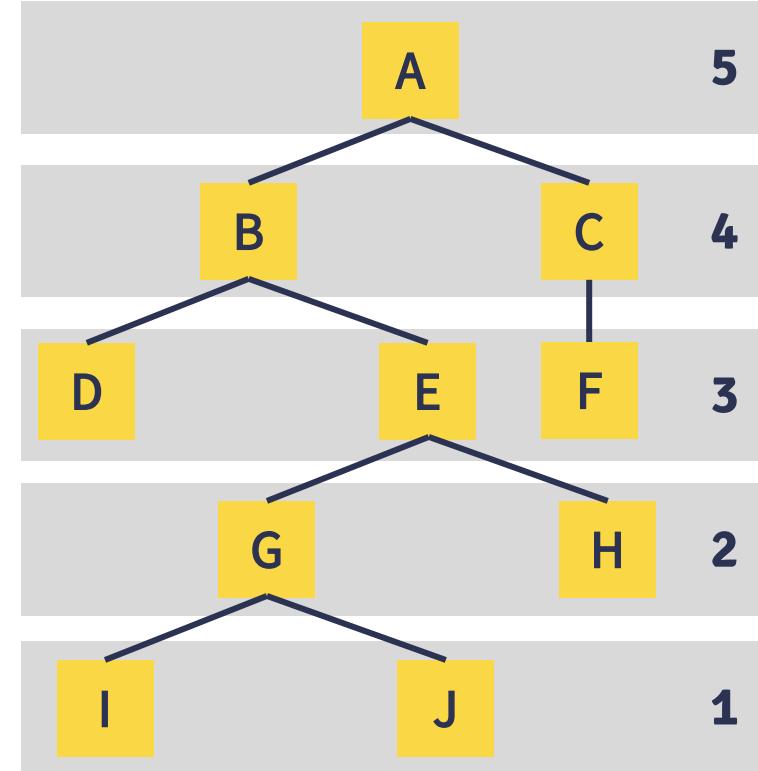
- **Sibling (Saudara)** : adalah node-node yang memiliki parent yang sama.



Sibling (D) = E

Istilah & Hubungan Komponen Tree ?

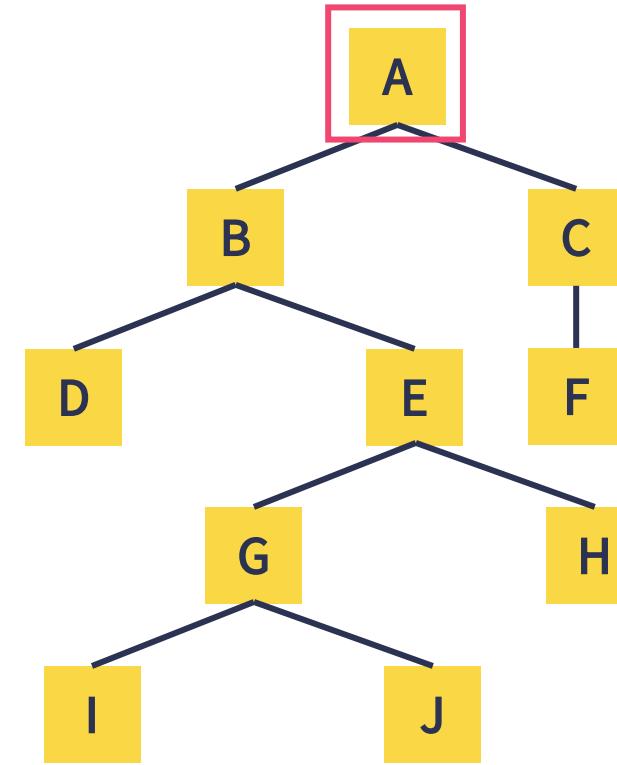
- **Height** : Banyaknya tingkatan dalam suatu tree.



Height = 5

Istilah & Hubungan Komponen Tree ?

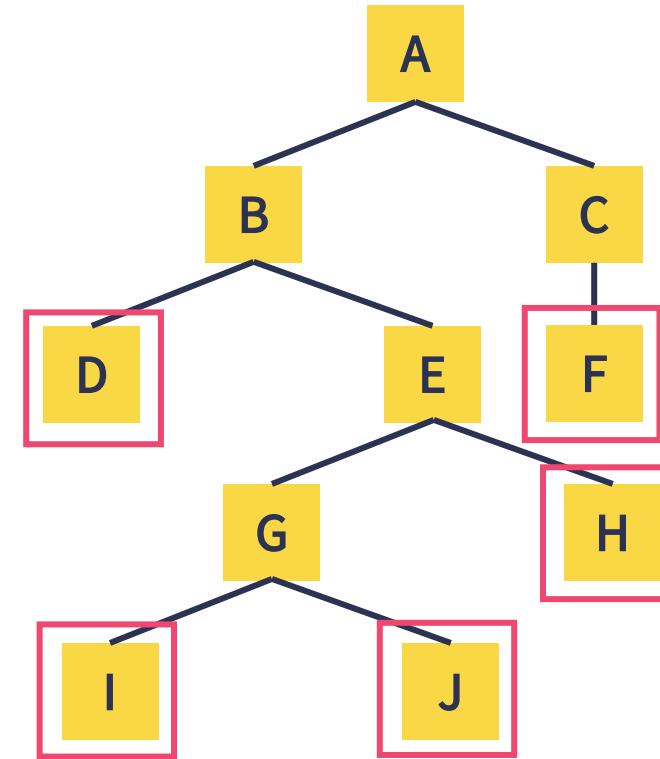
- **Root (Akar)** : adalah node khusus yang tidak memiliki predecessor (pendahulu).



Root = A

Istilah & Hubungan Komponen Tree ?

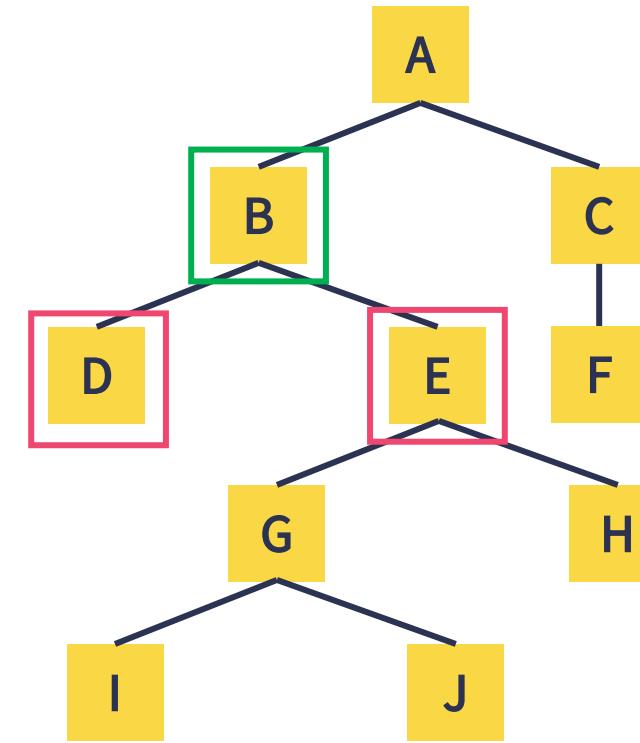
- **Leaf (Daun)** : adalah node-node dalam tree yang tidak memiliki successor (penerus).



Leaf = D,F,H,I,J

Istilah & Hubungan Komponen Tree ?

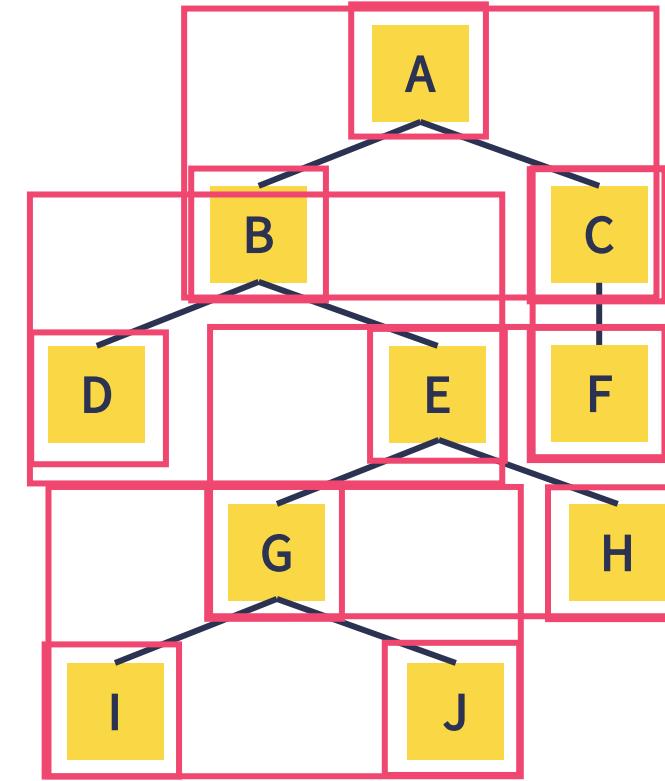
- **Degree** : adalah banyaknya child (anak) dalam suatu node.



Degree (B) = 2

Istilah & Hubungan Komponen Tree ?

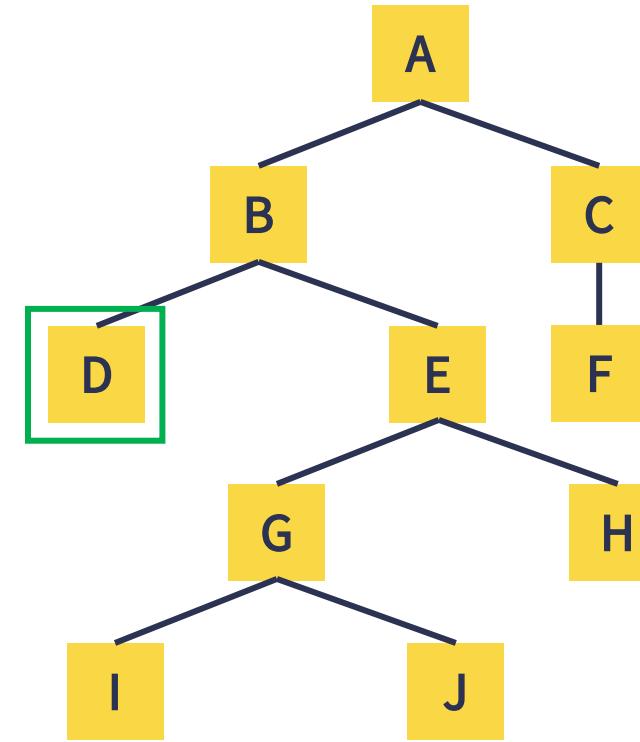
- **Forest (Hutan)** : adalah kumpulan dari tree.



Forest = 15

Istilah & Hubungan Komponen Tree ?

- **Depth (Kedalaman)** : adalah hasil tingkat node maksimum dikurang satu (level dari node x).



Depth (D) = 2

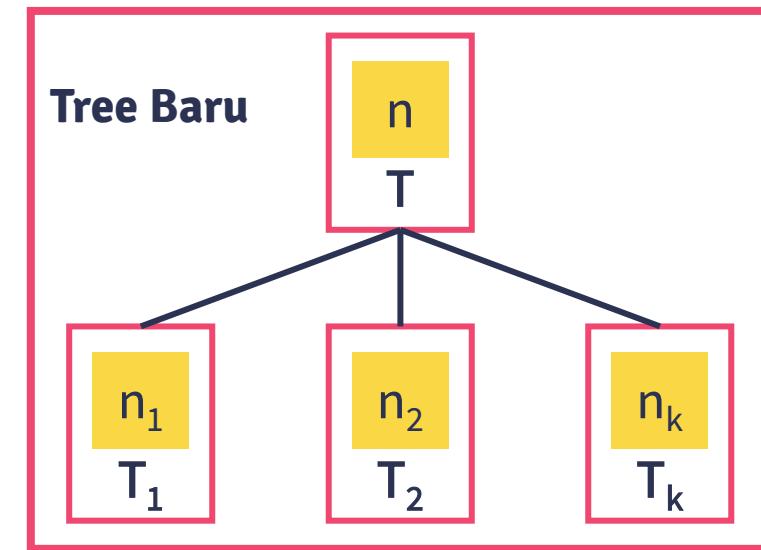
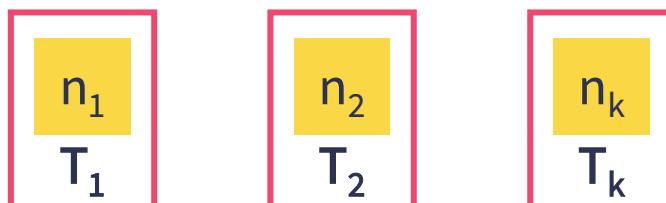
Definisi Tree ?

- Sebuah tree didefinisikan sebagai struktur yang dibentuk secara rekursif oleh aturan berikut.
 - Sebuah node adalah sebuah tree. Node satu-satunya pada tree ini berfungsi sebagai root maupun leaf.
 - Dari k buah tree $T_1 \sim T_k$, dan masing-masing memiliki root $n_1 \sim n_k$.
 - Jika node n adalah parent dari $n_1 \sim n_k$, akan diperoleh sebuah tree baru T yang memiliki root n. Dalam kondisi ini, tree $T_1 \sim T_k$ menjadi subtree dari tree T.

T : Tree

N : Node

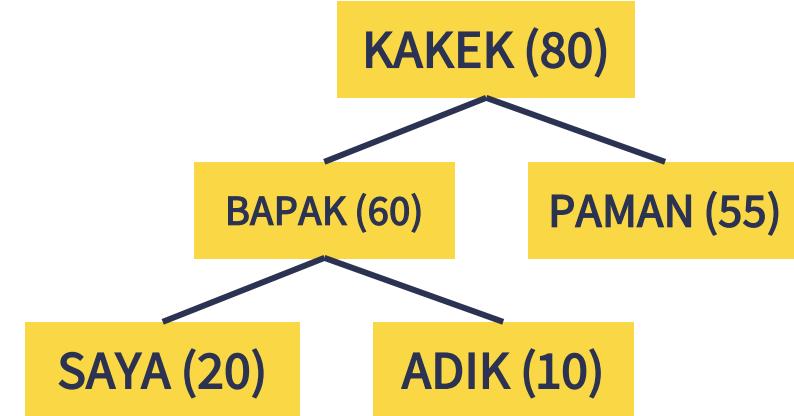
k : banyak



Ordered & Unordered Tree ?

- Ordered Tree

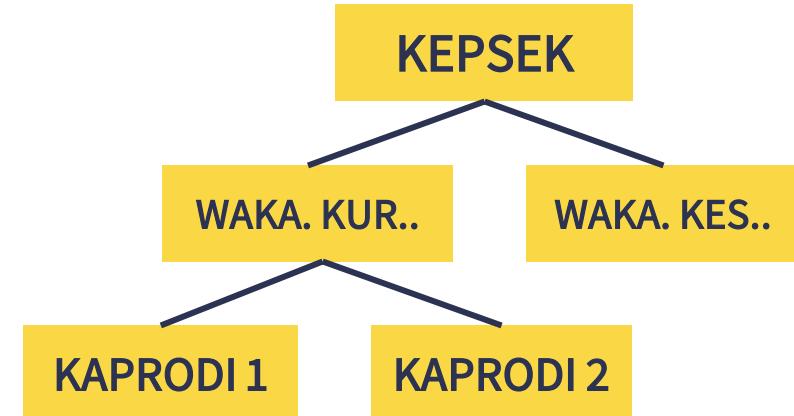
- Antar sibling (saudara) terdapat urutan “usia”.
- Node yang paling kiri berusia paling tua (sulung), sedangkan node yang paling kanan berusia paling muda (bungsu).
- Posisi node diatur atas urutan tertentu.
- Contoh : silsilah keluarga.



Ordered Tree

- Unordered Tree

- Antar sibling (saudara) tidak terdapat urutan tertentu.
- Contoh : struktur organisasi sekolah SMK.

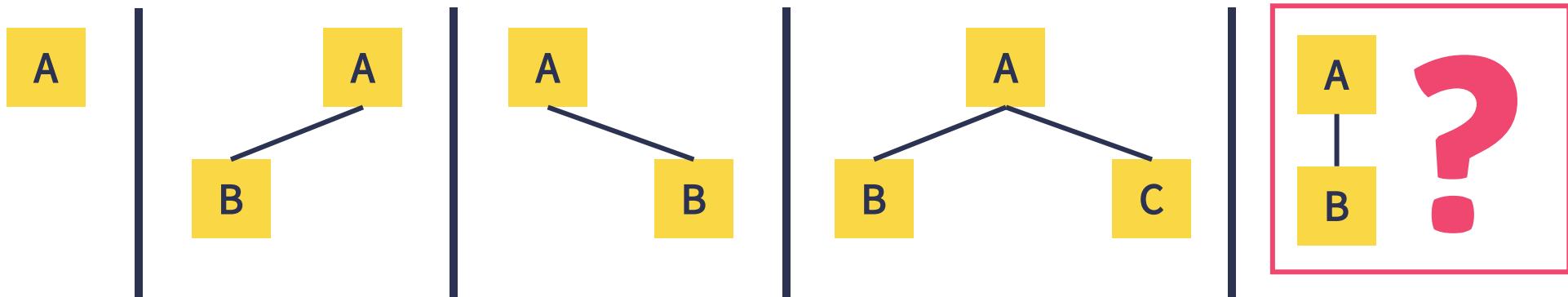


Unordered Tree

Konsep Binary Tree ?

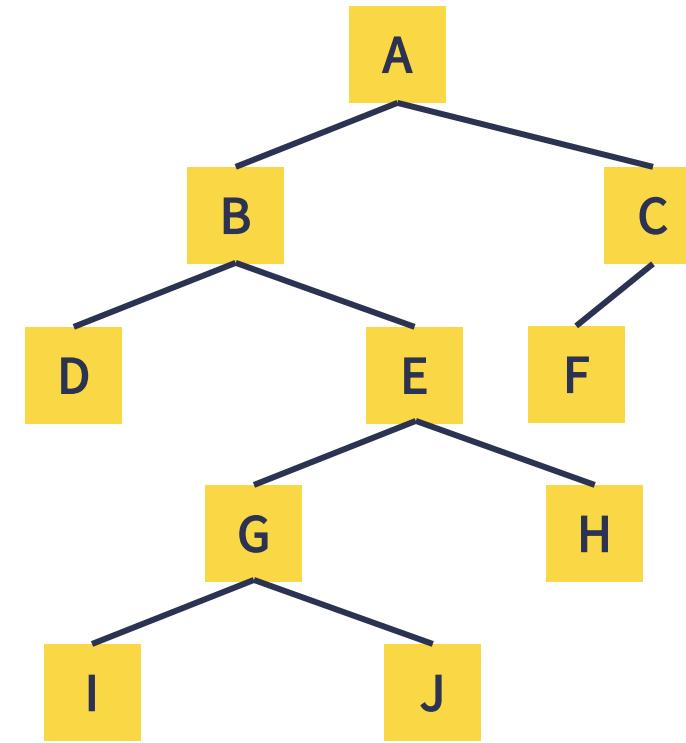
- Binary adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree harus terpisah.
 - Binary tree boleh tidak memiliki child (anak) ataupun subtree.
 - Boleh hanya memiliki subtree sebalah kiri (left subtree).
 - Boleh hanya memiliki subtree sebalah kanan (right subtree).
 - Boleh hanya memiliki subtree sebalah kiri (left subtree) dan kanan (right subtree).

Hati-hati menggambarkan Binary Tree



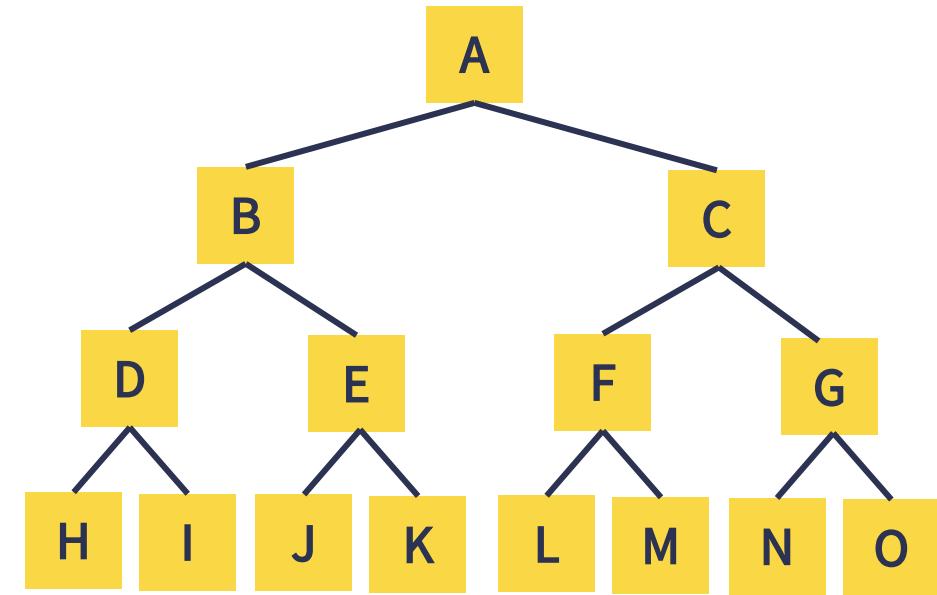
Jenis-jenis Binary Tree ?

- Full Binary Tree
- Complete Binary Tree
- Skewed Binary Tree



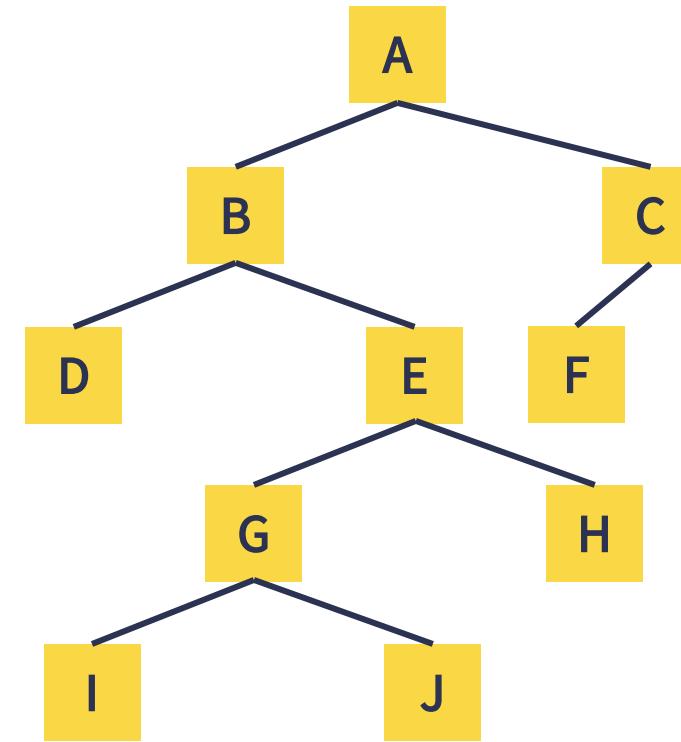
Jenis-jenis Binary Tree ?

- **Full Binary Tree :** adalah binary tree yang tiap node nya (kecuali leaf) memiliki dua child dan tiap subtree mempunyai panjang patch yang sama.



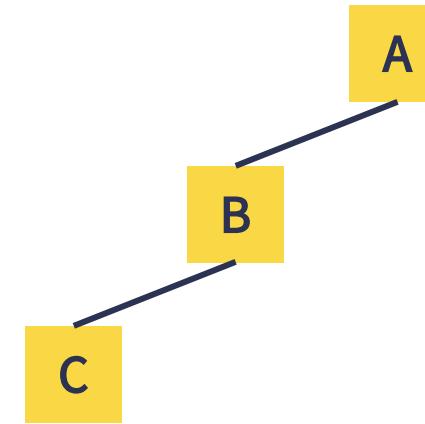
Jenis-jenis Binary Tree ?

- **Complete Binary Tree :** adalah binary tree yang mirip dengan full binary tree, namun setiap subtree boleh memiliki panjang patch yang berbeda.



Jenis-jenis Binary Tree ?

- **Skewed Binary Tree** : adalah binary tree yang semua node nya (kecuali left) hanya memiliki satu child.



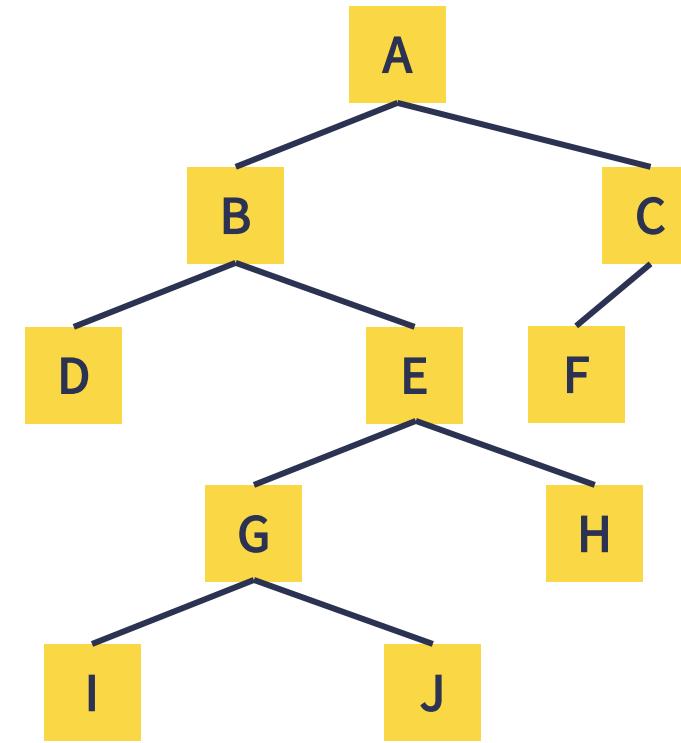
Definisi Tree Tranversal ?

- Adalah teknik menyusuri tiap node dalam sebuah tree secara sistematis, sehingga semua node dapat dan hanya satu kali saja dikunjungi.
- Ada tiga cara tranversal :
 - preOrder.
 - inOrder.
 - postOrder.
- Untuk tree atau node yang kosong, tranversal tidak perlu dilakukan.

Cara Tranversal ?

preOrder

1. Kunjungi root nya.
2. Telusuri subtree kiri.
3. Telusuri subtree kanan.

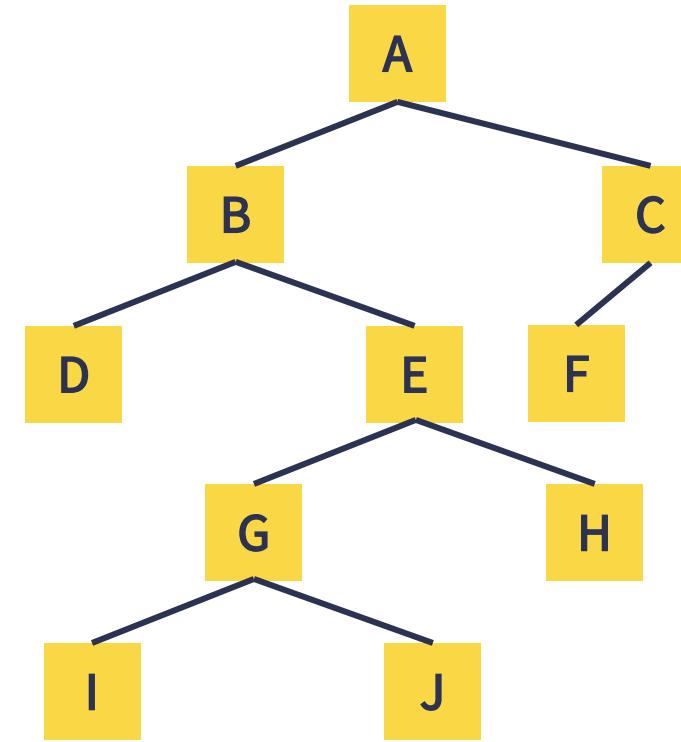


preOrder = A,B,D,E,G,I,J,H,C,F

Cara Tranversal ?

inOrder

1. Telusuri subtree kiri.
2. Kunjungi root nya.
3. Telusuri subtree kanan.

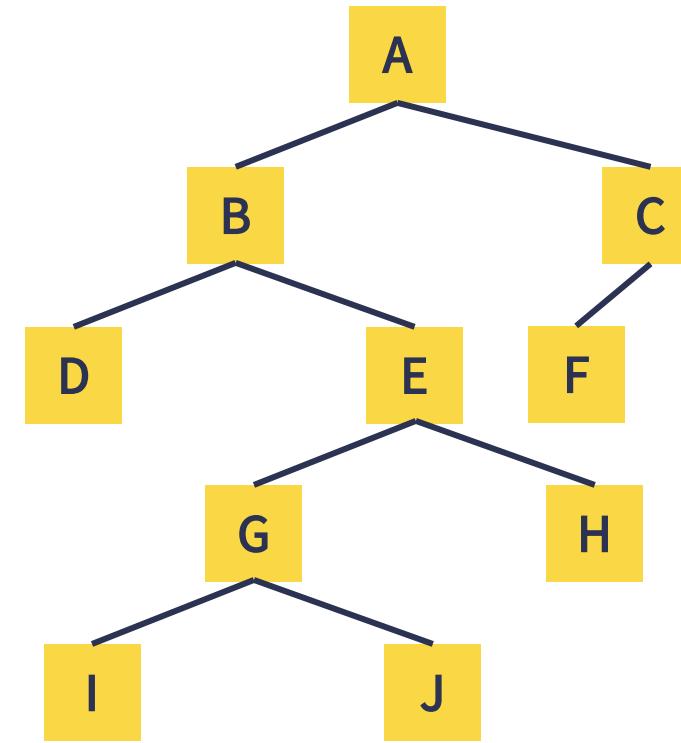


inOrder = D,B,I,G,J,E,H,A,F,C

Cara Tranversal ?

postOrder

1. Telusuri subtree kiri.
2. Telusuri subtree kanan.
3. Kunjungi root nya.



postOrder = D,I,J,G,H,E,B,F,C,A

Operasi pada Tree ?

- **Create** : digunakan untuk membentuk binary tree baru yang masih kosong.
- **Clear** : digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- **Empty** : digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- **Insert** : digunakan untuk memasukkan sebuah node kedalam tree.
- **Find** : digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- **Update** : digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- **Retrieve** : digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Delete Sub** : digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Characteristic** : digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average length-nya.
- **Traverse** : digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal.



Video Selanjutnya

Implementasi Tree C++



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE **8P**

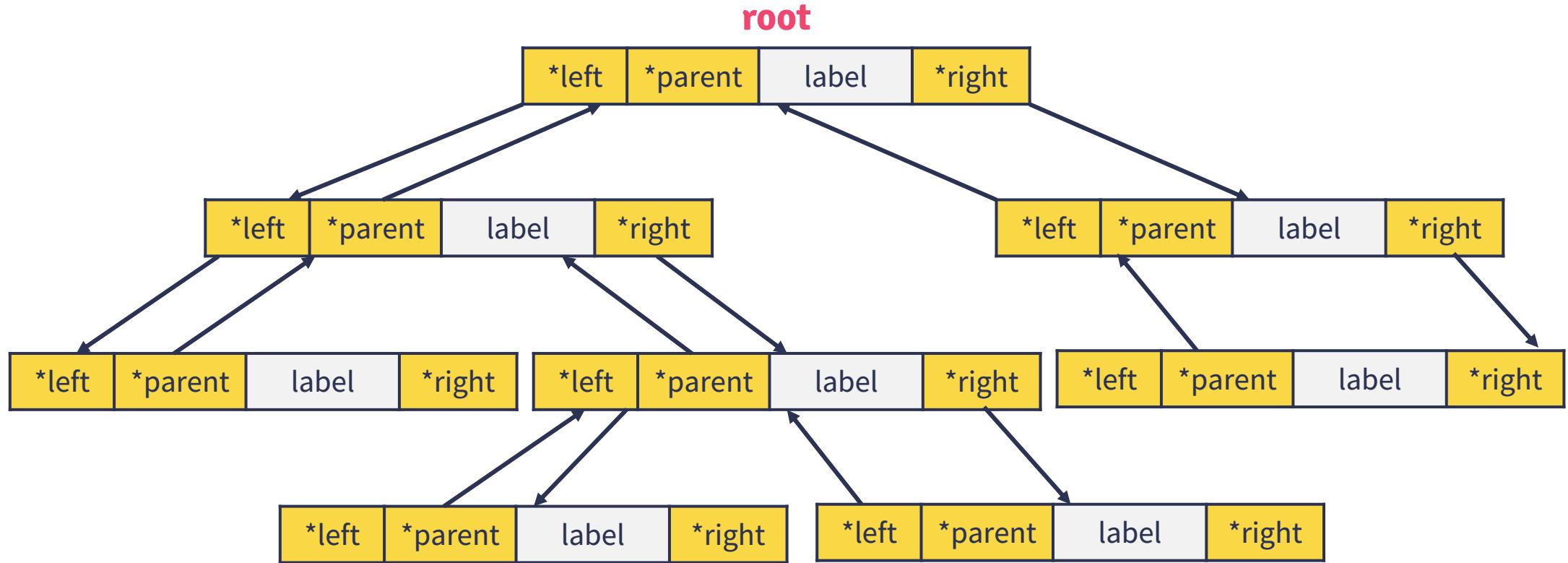
Implementasi Binary Tree C++

Pra-Syarat & Materi yang Dipakai ?

- Dasar Pemrograman C++
- Fungsi pada C++
- Pointer
- Struct (struktur)
- Linked List



Konsep Implementasi Binary Tree ?



- Masing-masing node dibuat menggunakan linked list.
- Terdapat pointer left (menunjuk ke anak kiri), right (menunjuk ke anak kanan) dan parent (menunjuk ke node orang tua).
- Pointer parent pada node root NULL karena root tidak memiliki orang tua.
- Pointer parent dapat membantu proses saat mengetahui parent dan sibling.

Operasi pada Tree ?

- **Create** : digunakan untuk membentuk binary tree baru yang masih kosong.
- **Clear** : digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- **Empty** : digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- **Insert** : digunakan untuk memasukkan sebuah node kedalam tree.
- **Find** : digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- **Update** : digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- **Retrieve** : digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Delete Sub** : digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Characteristic** : digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average length-nya.
- **Traverse** : digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal.

Cara Tranversal ?

preOrder

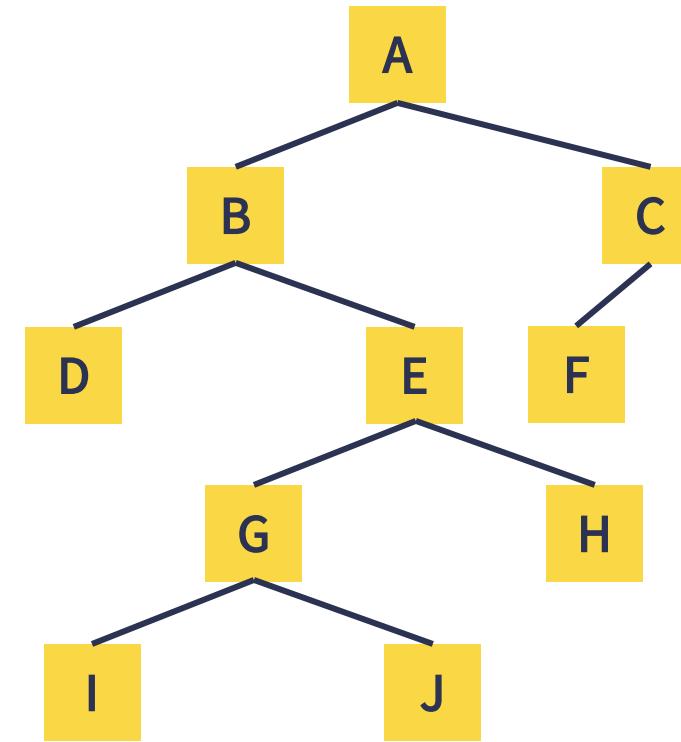
1. Kunjungi root nya.
2. Telusuri subtree kiri.
3. Telusuri subtree kanan.

inOrder

1. Telusuri subtree kiri.
2. Kunjungi root nya.
3. Telusuri subtree kanan.

postOrder

1. Telusuri subtree kiri.
2. Telusuri subtree kanan.
3. Kunjungi root nya.



preOrder = A,B,D,E,G,I,J,H,C,F

inOrder = D,B,I,G,J,E,H,A,F,C

postOrder = D,I,J,G,H,E,B,F,C,A



Video Selanjutnya

Konsep Graph



Thank you

#KEEPLEARNING
#KEEPSPIRITS





Struktur Data

Saniati,S.ST.,M.T.

EPISODE 9

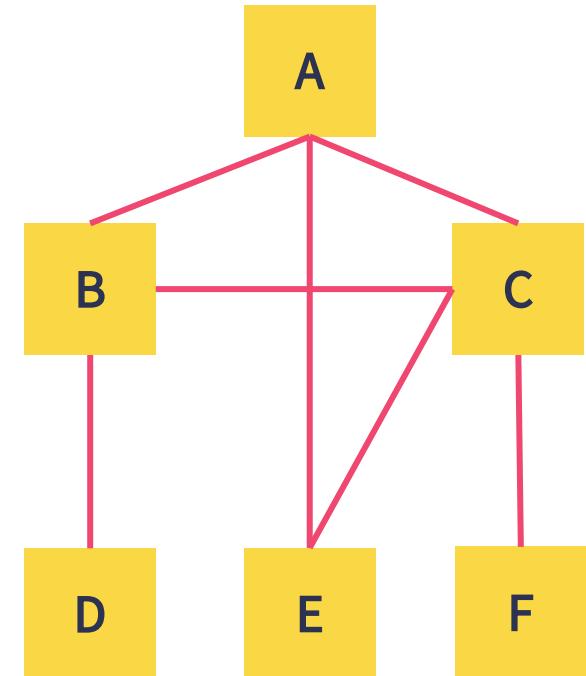
Graph

Yang dipelajari ?

- Konsep Graph
- Contoh Penggunaan Graph
- Masalah-masalah Graph
- Struktur Data Linear **vs** Tree **vs** Graph
- Undirected, Directed & Weighted Graph
- Graph Path
- Operasi Graph
- Adjacency Matrices dan Adjacency List

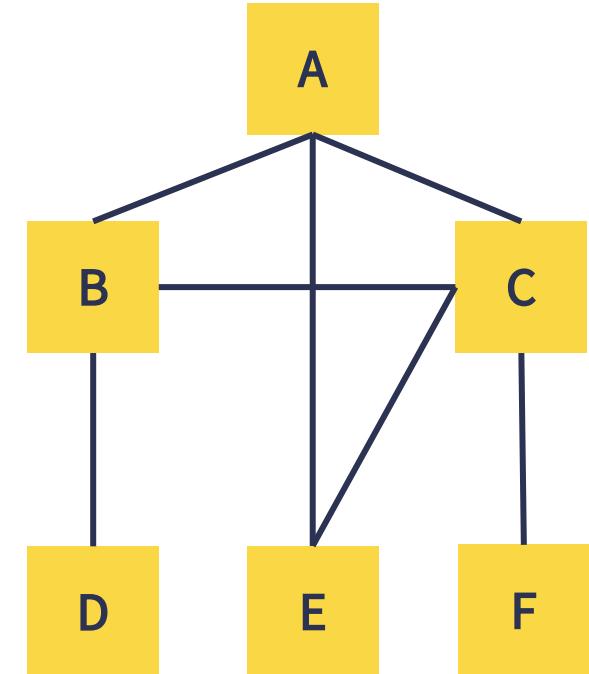
Konsep Graph ?

- Konsep struktur data yang terdiri dari **node** (vertex atau vertices) dan **garis penghubung** (arc atau edge).
- **Vertex** disimbolkan dengan “**V**” dan **edge** disimbolkan dengan “**E**”.
- Keterhubungan graph dapat membentuk relasi One-to-One, One-to-Many, Many-to-One dan Many-to-Many.
- Contoh : Informasi topologi jaringan, keterhubungan antar kota-kota, dll.



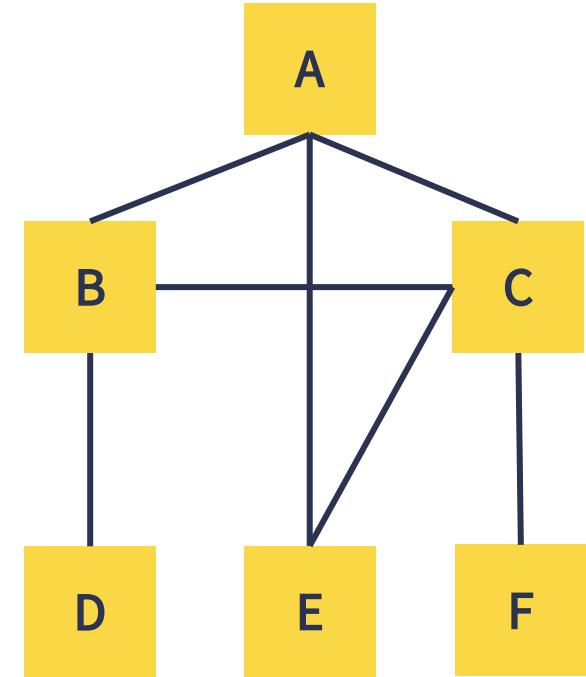
Contoh Penggunaan Graph ?

- Graph banyak digunakan untuk :
 - Menggambarkan jaringan dan peta jalan, jalan kereta api, lintasan pesawat, system perpipaan, saluran telepon, koneksi elektrik, ketergantungan diantara task pada sistem manufaktur dan lain-lain.
- Terdapat banyak hasil dan struktur penting yang didapatkan dari perhitungan dengan graph.



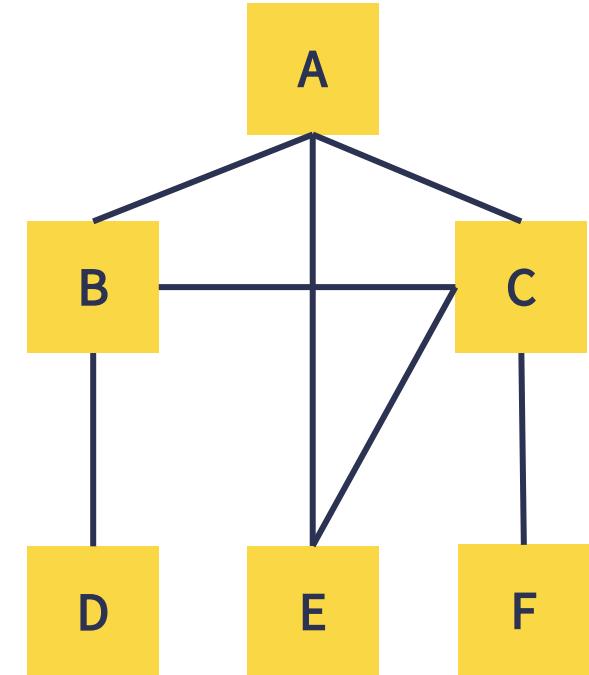
Masalah-masalah Graph ?

- **Shortest Path Problem** (Masalah Patch Minimum) : mencari route dengan jarak terpendek dalam suatu jaringan transportasi.
- **Maximum Flow Problem** (Masalah Aliran Maksimum) : menghitung volume aliran BBM dari suatu reservoir ke suatu titik tujuan melalui jaringan pipa.
- **Graph Searching Problem** (Masalah Pencarian dalam Graph) : mencari langkah-langkah terbaik dalam program permainan catur komputer.
- **Topological Ordering Problem** (Masalah Pengurutan Topologi) : menentukan urutan pengambilan mata-mata kuliah yang saling berkaitan dalam hubungan prasyarat (prerequisite).
- **Task Network Problem** (Masalah Jaringan Tugas) : membuat penjadwalan penggerjaan suatu proyek yang memungkinkan waktu penyelesaian tersingkat.



Masalah-masalah Graph ?

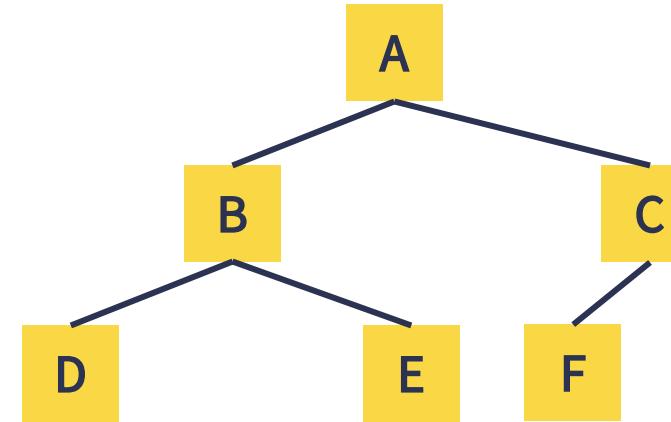
- **Minimum Spanning Tree Problem** (Masalah Pencarian Pohon Rentang Minimum) : mencari rentangan kabel listrik yang totalnya adalah minimal untuk menghubungkan sejumlah kota.
- **Travelling Salesperson Problem** : tukang pos mencari lintasan terpendek melalui semua alamat penerima pos tanpa harus mendatangi suatu tempat lebih dari satu kali.
- **Four-color Problem** : dalam menggambar peta, memberikan warna yang berbeda pada setiap propinsi yang saling bersebelahan.



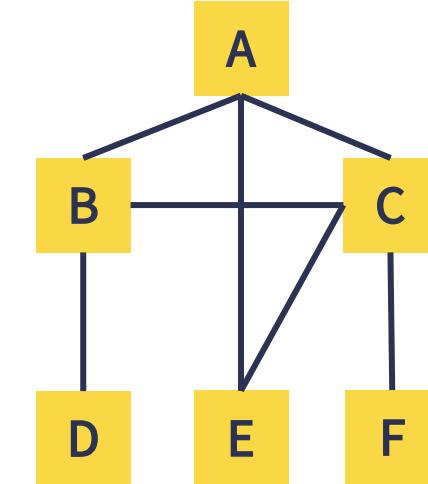
Struktur Data Linear vs Tree vs Graph ?



Linear



Tree

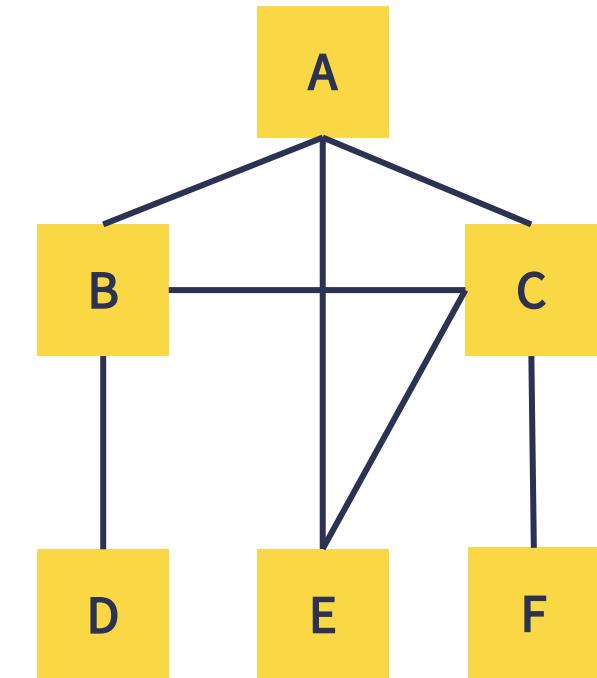


Graph

- **Struktur Data Linear** : keterhubungan sekuel (terurut) antara entitas data.
- **Struktur Data Tree** : keterhubungan hirarkis.
- **Struktur Data Graph** : keterhubungan tak terbatas antara entitas data.

Kategori Graph ?

- Undirected Graph
- Directed Graph
- Weighted Graph



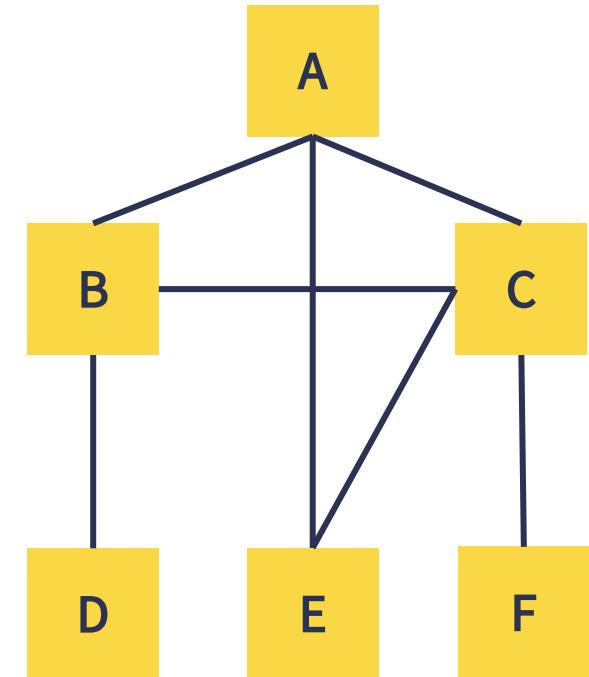
Undirected Graph (Undigraph) ?

- Edge tidak ada penugasan arah dari satu vertex ke vertex yang lain.
- Digunakan untuk mempresentasikan relasi One-to-One.
- Jika A dan B adalah vertex, maka edge dapat mempresentasikan sebagai (A,B) dan (B,A).

Contoh pengkerjaan Undirected Graph :

Vertex : A, B, C, D, E, F

**Edges : (A,B), (A,C), (A,E), (B,A), (B,C), (B,D), (C,A),
(C,B), (C,E), (C,F), (D,B), (E,A), (E,C), (F,C)**



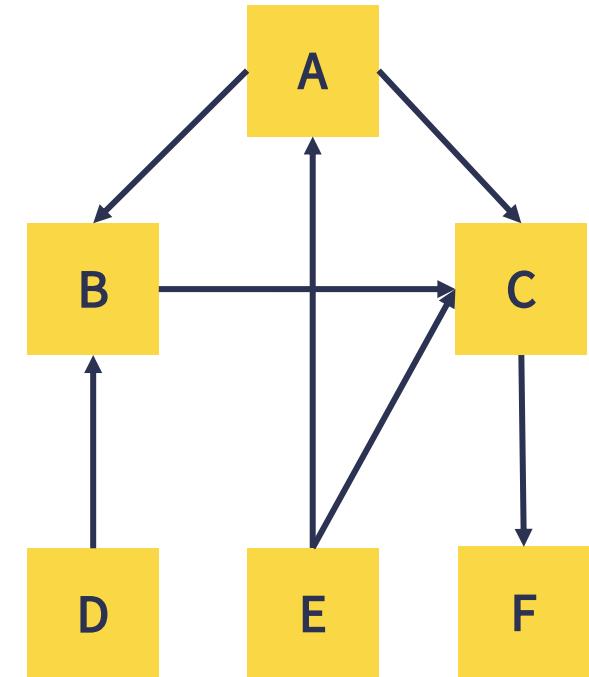
Directed Graph (Digraph) ?

- Edge ada penugasan arah dari satu vertex ke vertex yang lain.
- Edges antar Vertexts diwakilkan dengan kurung sudut.
- Jika vertex A sebagai sumber dan menunju ke vertex B sebagai tujuan, maka edge dapat mempresentasikan sebagai $\langle A, B \rangle$.

Contoh penggerjaan Directed Graph :

Vertex : A, B, C, D, E, F

Edges : $\langle A, B \rangle$, $\langle A, C \rangle$, $\langle B, C \rangle$, $\langle C, F \rangle$, $\langle D, B \rangle$, $\langle E, A \rangle$, $\langle E, C \rangle$



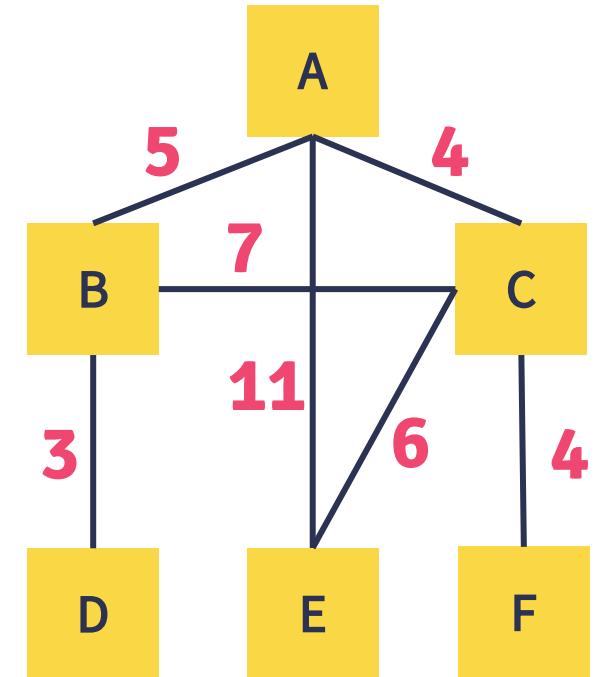
Weighted Graph ?

- Edge diberikan suatu nilai sebagai weight dari satu vertex ke vertex yang lain.
- Dapat direpresentasikan oleh Undirected atau Directed Graph.
- Weight dapat direpresentasikan sebagai jarak, lama waktu, dll.

Contoh pengajaran Undirected Weight Graph :

Vertex : A, B, C, D, E, F

Edges : $(A,B) = 5$, $(A,C) = 4$, $(A,E) = 11$, $(B,A) = 5$, $(B,C) = 7$,
 $(B,D) = 3$, $(C,A) = 4$, $(C,B) = 7$, $(C,E) = 6$, $(C,F) = 4$, $(D,B) = 3$,
 $(E,A) = 11$, $(E,C) = 6$, $(F,C) = 4$



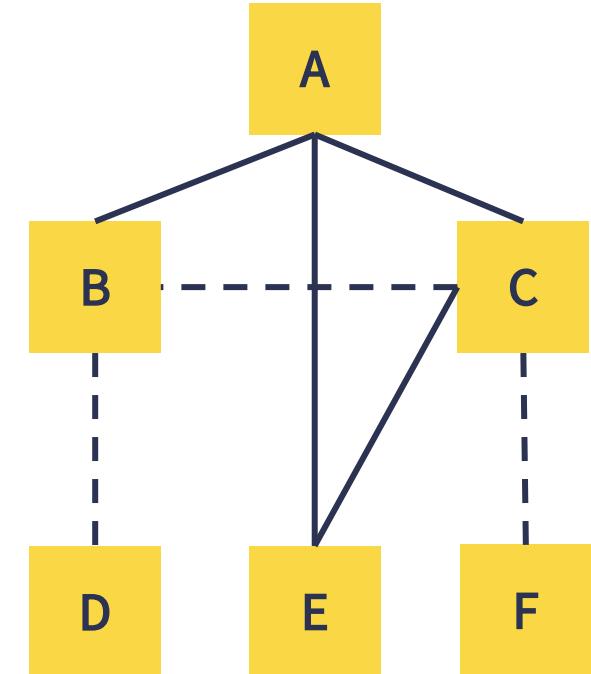
Graph Paths ?

- Path adalah urutan vertice yang dihubungkan oleh edges atau adjacent vertices.

Contoh pengajaran Graph Paths :

Paths dari Node D ke Node F : D, B, C, F

Paths D B C F yaitu 4



Operasi pada Graph ?

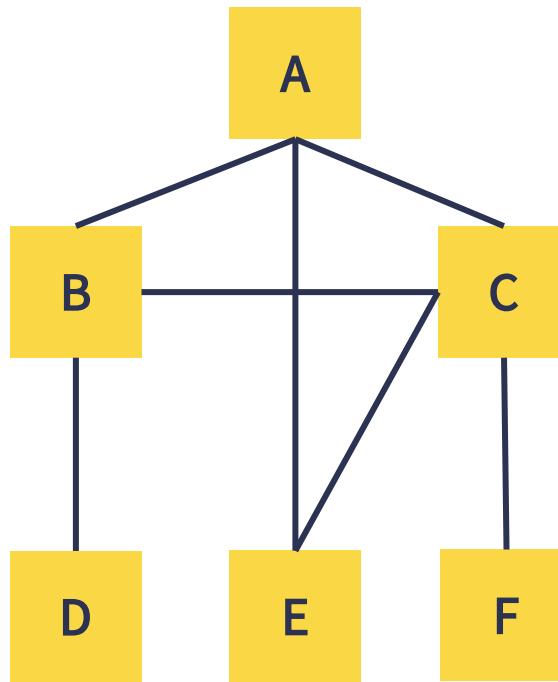
- **Add Vertex** : menambah vertex baru ke dalam graph.
- **Delete Vertex** : menghapus vertex tertentu disuatu graph.
- **Add Edge** : menambah edge baru yang menghubungkan antar vertex disuatu graph.
- **Delete Edge** : menghapus suatu edge yang menghubungkan antar vertex disuatu graph.
- **Search Path** : mencari panjang paths dari suatu vertex ke vertex yang lain.
- **Traversing** : penelusuran dari vertex ke vertex lainnya dengan penghubung edge.

Konsep Implementasi Graph ?

- **Adjacency Matrices (Array Based)** : implementasi menggunakan array matriks.
- **Adjacency List (Linked List)** : implementasi menggunakan linked list.

Adjacency Matrices (Array Based) ?

- Representasinya dengan array 2 dimensi atau matriks.
- Pada Undirect & Direct Graph 0 berarti tidak ada relasi edge, sedangkan nilai 1 ada relasi edge.
- Pada Weighted Graph nilainya diisi nilai beban edge nya.

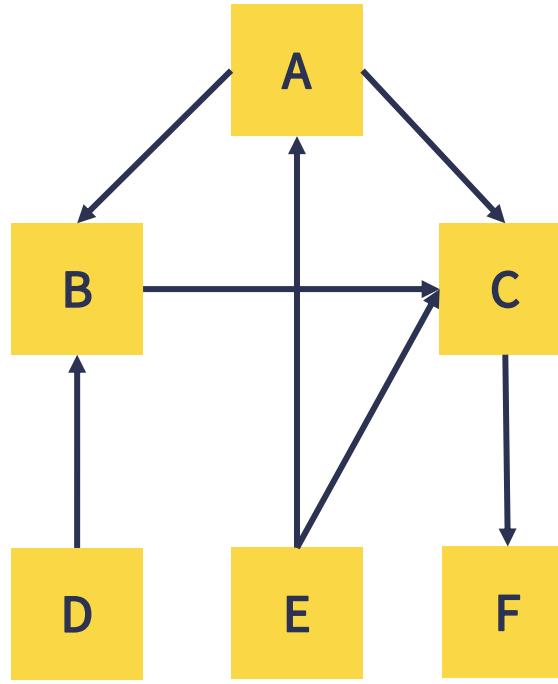


		Tujuan					
		A	B	C	D	E	F
Sumber	A	0	1	1	0	1	0
	B	1	0	1	1	0	0
	C	1	1	0	0	1	1
	D	0	1	0	0	0	0
	E	1	0	1	0	0	0
	F	0	0	1	0	0	0

Undirect Graph

Adjacency Matrices (Array Based) ?

- Representasinya dengan array 2 dimensi atau matriks.
- Pada Undirect & Direct Graph 0 berarti tidak ada relasi edge, sedangkan nilai 1 ada relasi edge.
- Pada Weighted Graph nilainya diisi nilai beban edge nya.

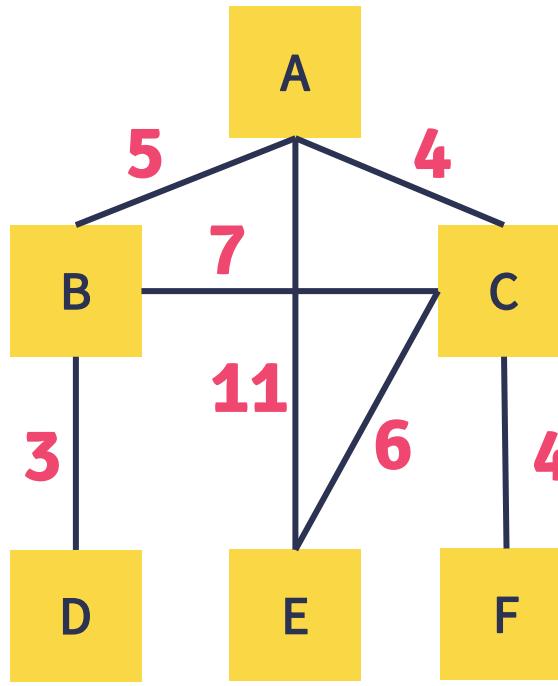


		Tujuan					
		A	B	C	D	E	F
Sumber	A	0	1	1	0	0	0
	B	0	0	1	0	0	0
	C	0	0	0	0	0	1
	D	0	1	0	0	0	0
	E	1	0	1	0	0	0
	F	0	0	0	0	0	0

Direct Graph

Adjacency Matrices (Array Based) ?

- Representasinya dengan array 2 dimensi atau matriks.
- Pada Undirect & Direct Graph 0 berarti tidak ada relasi edge, sedangkan nilai 1 ada relasi edge.
- Pada Weighted Graph nilainya diisi nilai beban edge nya.



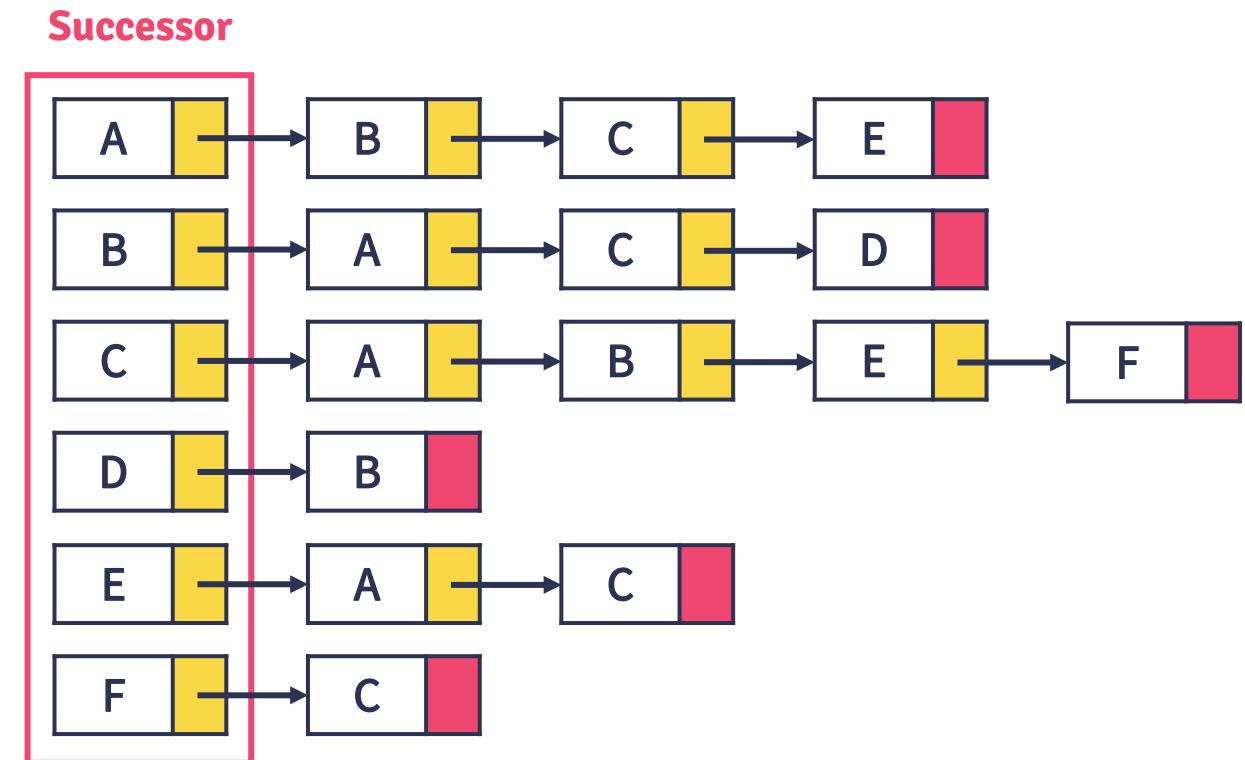
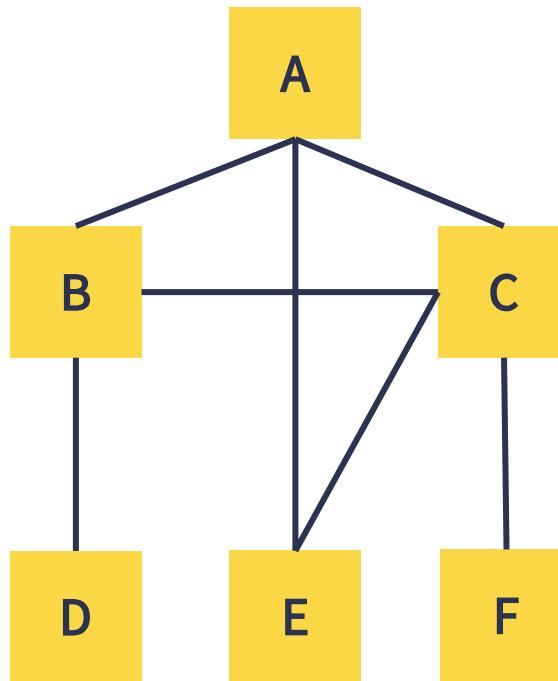
Tujuan

Sumber	A	B	C	D	E	F
A	0	5	4	0	11	0
B	5	0	7	3	0	0
C	4	7	0	0	6	4
D	0	3	0	0	0	0
E	11	0	6	0	0	0
F	0	0	4	0	0	0

Undirect Weighted Graph

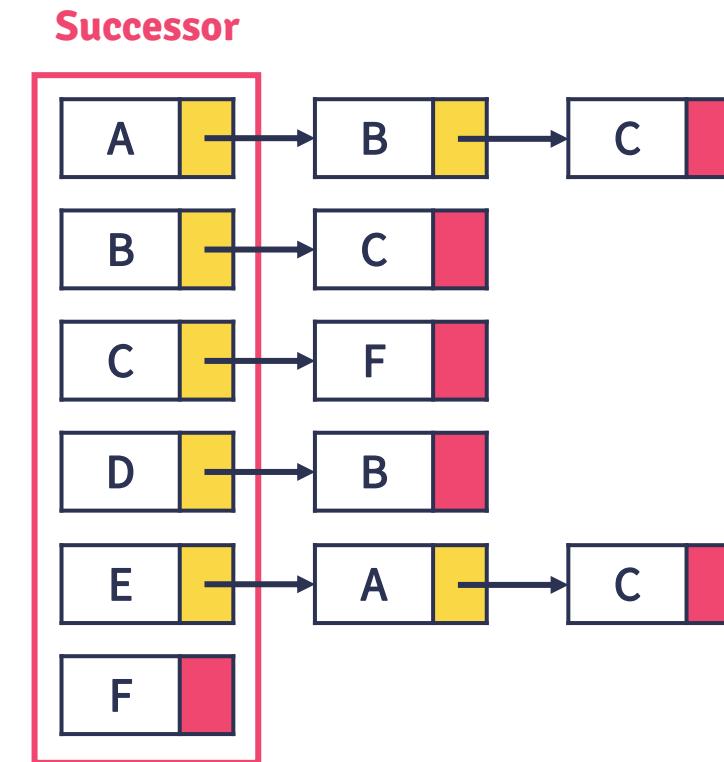
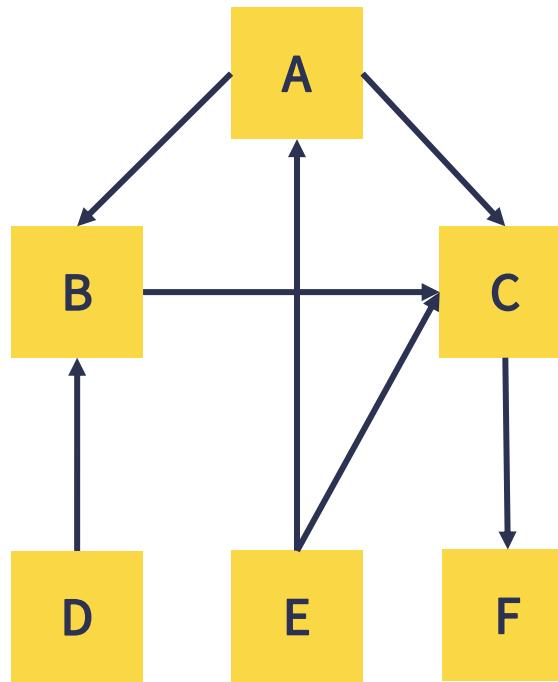
Adjacency List (Linked List) ?

- Representasinya dengan linked list.
- Node asal akan disebut Successor.
- Isi dari successor akan diisi list node yang berkaitan, node terakhir akan diisi pada nextnya.
- Implementasinya harus paham PBO (Pemrograman Berbasis Objek).



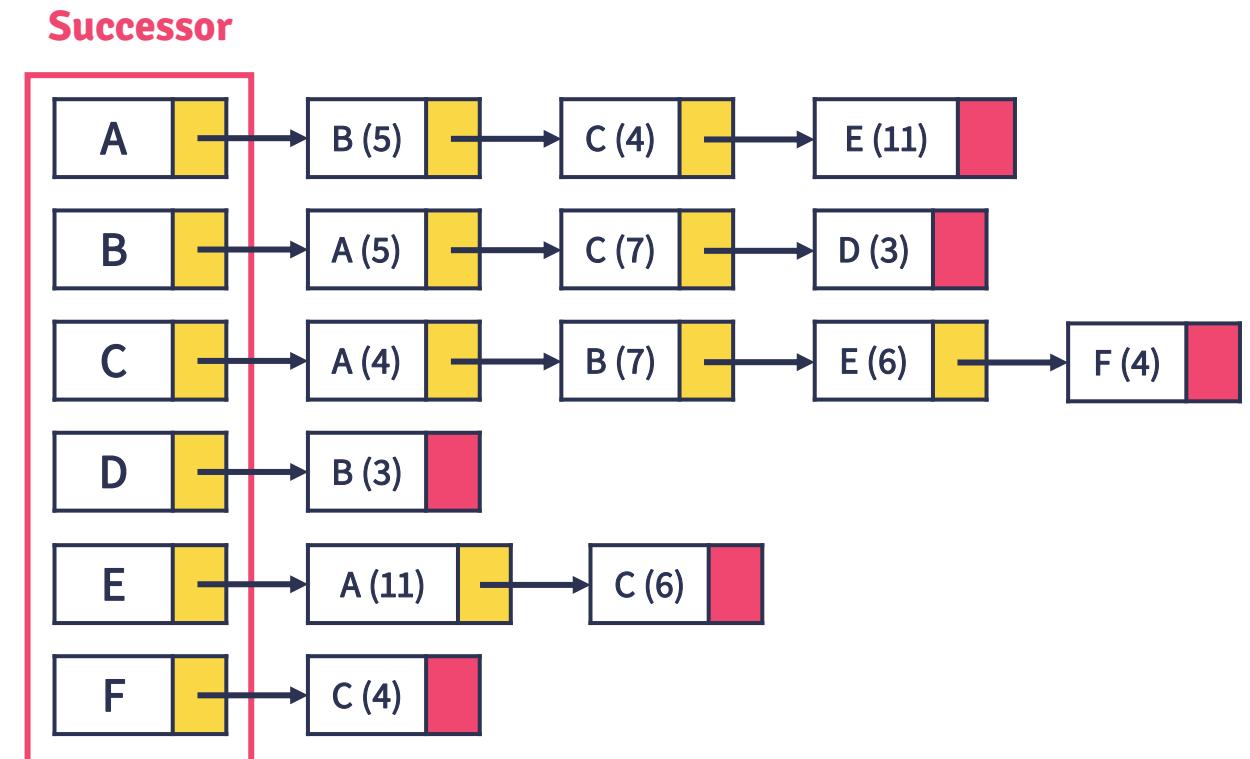
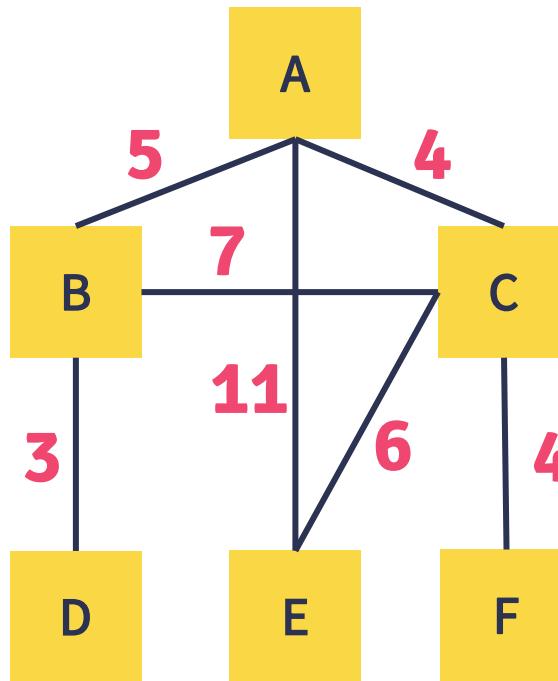
Adjacency List (Linked List) ?

- Representasinya dengan linked list.
- Node asal akan disebut Successor.
- Isi dari successor akan diisi list node yang berkaitan, node terakhir akan diisi pada nextnya.
- Implementasinya harus paham PBO (Pemrograman Berbasis Objek).



Adjacency List (Linked List) ?

- Representasinya dengan linked list.
- Node asal akan disebut Successor.
- Isi dari successor akan diisi list node yang berkaitan, node terakhir akan diisi pada nextnya.
- Implementasinya harus paham PBO (Pemrograman Berbasis Objek).





Video Selanjutnya

Selesai





Thank you

#KEEPLEARNING
#KEEPSPIRITS





Sistem Kebut Semalam

Yunus Febriansyah

EPISODE 2

BELAJAR DASAR C++

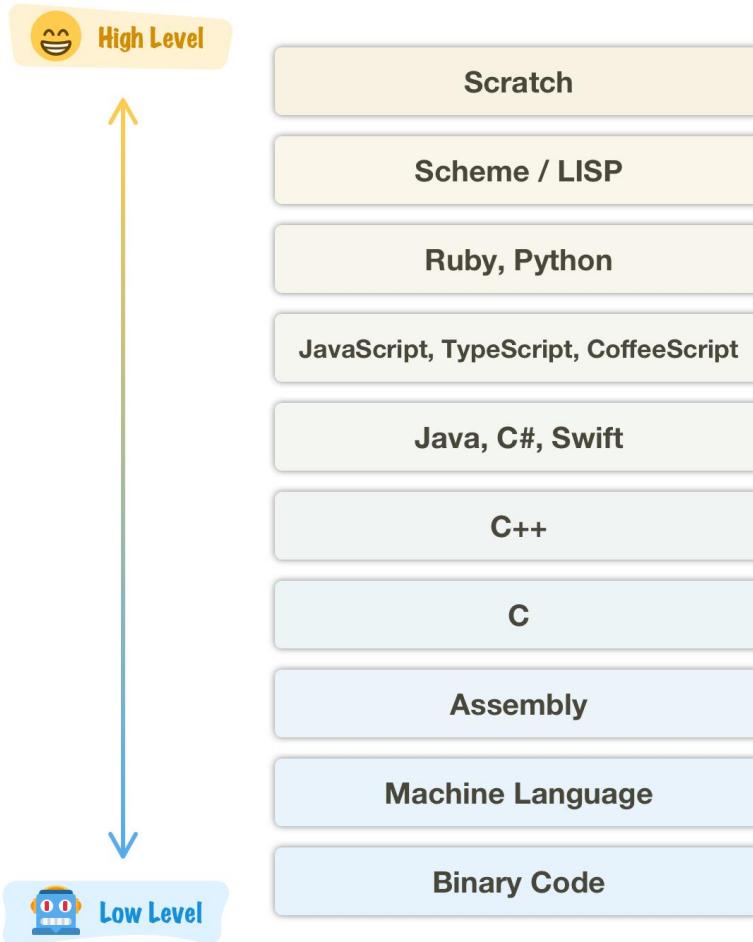
Sejarah Singkat ?

- Bahasa C++ lahir pada tahun 1980, yang dibuat oleh Bjarne Stroustrup di AT&T Bell Laboratories awal tahun 1980-an berdasarkan C ANSI (American National Standard Institute). Pertama kali, prototype C++ muncul sebagai C yang dipercanggih dengan fasilitas kelas, bahasa tersebut disebut "C dengan kelas" (C with Class).
- Untuk mendukung fitur-fitur pada C++, dibangun efisiensi dan sistem support untuk pemrograman tingkat rendah (low level coding). Pada C++ ditambahkan konsep-konsep baru seperti class dengan sifat-sifatnya seperti inheritance dan overloading.[butuh rujukan] Salah satu perbedaan yang paling mendasar dengan bahasa C adalah dukungan terhadap konsep pemrograman berorientasi objek (object-oriented programming).
- Biodata biografi.
https://en.wikipedia.org/wiki/Bjarne_Stroustrup



Source :<https://www.stroustrup.com/>

Low & High Level Programming Lang ?



Standarisasi C++ ?

Tahun	C++ Standar	Nama Informal
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11, C++0x
2014	ISO/IEC 14882:2014	C++14, C++1y
2017	ISO/IEC 14882:2017	C++17, C++1z
2020	Belum nemu	C++20,[17] C++2a

C, C++, C# ?



A yellow vertical bar on the left side of the slide features three circular icons: a power button icon at the top, a document icon in the middle, and a coffee cup icon at the bottom.

C



C++

C++



Contoh Aplikasi dengan C++ ?

- Winamp Media Player
- MySQL Server
- Mozilla Firefox
- Google Chrome
- Microsoft Office
- Adobe Photoshop & Illustrator
- Inkscape
- dan masih banyak lagi.

Kenapa C++ ?

- Bahasa C++ memiliki kapabilitas yang sangat baik sehingga programmer dapat memperoleh seluruh tenaga yang dimiliki komputer.
- Dapat dikembangkan di berbagai platform sehingga aplikasi yang dibangun dapat berjalan di sistem operasi yang berbeda.
- Compiler C++ yang sangat baik sehingga dapat mempercepat proses kompilasi.
- C++ merupakan bahasa terstruktur yang mendukung OOP(Object Oriented Programming).
- C++ dikategorikan sebagai bahasa tingkat menengah sehingga mendekati bahasa mesin.
- Dengan bahasa C++ kita dapat membangun aplikasi graphic processor menggunakan librari OpenGL.
- Kemudahan dalam memanipulasi data seperti merubah alamat dari suatu variabel menggunakan pointer

Dengan C++ ?

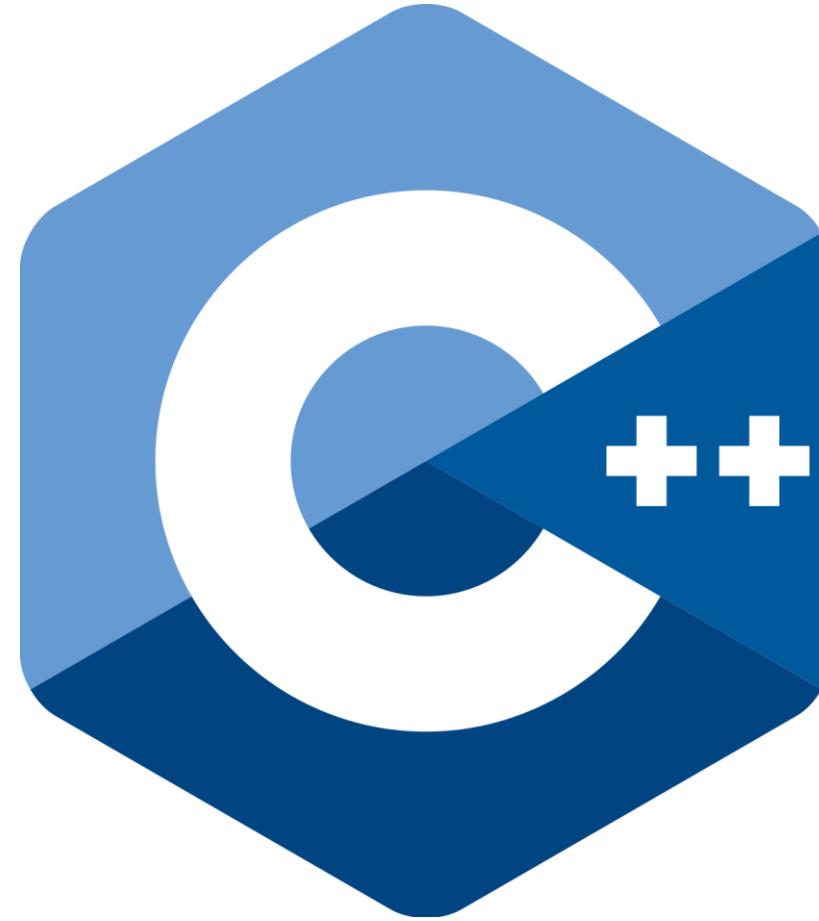
- Mempelajari Kernel dari Sistem Operasi
- Membangun aplikasi desktop
- Membuat aplikasi mikrokontroler
- Ikut mengembangkan teknologi open source
- Membuat library untuk bahasa pemrograman lain
- Membuat aplikasi perangkat mobile
- Membuat Game

Notes ?

- Compiler C++ yang banyak
- Banyak Bahasa pemrograman keturunan C

Materi ?

1. Pengantar
2. Persiapan
3. Pendahuluan
4. Comment
5. Variabel, Konstanta & Tipe Data
6. User Input
7. Operator
8. Casting
9. Percabangan IF
10. Percabangan Ternary
11. Percabangan Switch Case
12. For Loop
13. While Loop
14. Do.. While Loop
15. Array
16. Void Function
17. Return Function
18. Recursive Function
19. Variable Scope (Local, Global & Block)



Persiapan ?

* Text Editor



TURBO C++



Code::Blocks



Visual C++

Pendahuluan ?

```
#include <iostream>

int main()
{
    // code here

}
```



Comment ?

```
// Satu baris  
  
/*  
Komentar  
Banyak  
Baris  
*/
```



Variabel, Konstanta & Tipe Data ?

```
// Declare Variable
/*
    dataType varName;
    dataType varName = varValue;
    dataType varName1, varName2, varNameN;
    dataType varName1 = varVal1, varName2
        = varVal2;
*/
// Declare Constant
/*
    const dataType constName = constValue;
    const dataType constName1 =
        constVal1, constName2 = constVal2;
*/
```

```
// Bilangan Bulat
int Integer;
// Bilangan Pecahan
float Float;
// Bilangan Pecahan
double Double;
// Boolean
bool Boolean;
// String
string String;
```

User Input ?

```
// Declare Variable  
// dataType varName;  
  
// User Input  
// cin >> varName;  
  
// for String  
// getline(cin, varName)
```



Operator ?

// Aritmatika

// +, -, *, /, %, ++, --

// Pengisian / Assignment

// =, +=, -=, *=, /=, %=,
// ^=, &=, |=, >>=, <<=

// Pembanding

// ==, !=, >, <, >=, <=

// Logika

// &&, ||, !



Casting ?

```
// Berlaku untuk bilangan  
// (dataType) varName or Number
```



Percabangan IF ?



```
// IF Tunggal
/*
if (condition) {
    // block of code to be executed if the
    condition is true
}
*/
// IF Ganda
/*
if (condition) {
    // block of code to be executed if the
    condition is true
} else{
    // block of code to be executed if the
    condition is false
}
*/
```

```
// IF Majemuk
/*
if (condition1) {
    // block of code to be executed if the
    condition1 is true
} else if (condition2) {
    // block of code to be executed if the
    condition2 is true
} else if (conditionN) {
    // block of code to be executed if the
    conditionN is true
} else{
    // block of code to be executed if all
    condition is false
}
*/
```

Percabangan Ternary ?

```
// variable = (condition) ? expressionTrue : expressionFalse;
```



Percabangan Switch Case ?

```
// Switch Case
/*
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
*/
```



For Loop ?

```
// For Loop
/*
for (nilaiAwal; kondisi; step) {
    // Eksekusi
}
```



While Loop ?

```
// While Loop
/*
    while (condition) {
        // code block to be executed
    }
*/
```

Do..While Loop ?

```
// Do..While Loop
/*
do {
    // code block to be executed
}
while (condition);
*/
```

Array ?

```
// Declare
/*
    dataType arrName[arrSize];
    dataType arrName[arrSize] = {val1, valN};
*/
// Assignment
/*
    arrName[index] = value;
*/
```



Void Function ?

```
// Declare
/*
void funcName(){
    // code to be executed
}
*/
/*
void funcName(dataType par1, dataType parN){
    // code to be executed
}
*/

// Call
/*
funcName();
*/
/*
funcName(arg1, argN);
*/
```



Return Function ?

```
// Declare
/*
    dataType funcName(){
        // code to be executed
        return (value must match data type)
    }
*/
/*
    void funcName(dataType par1, dataType parN){
        // code to be executed
        return (value must match data type)
    }
*/

// Call
/*
    funcName();
*/
/*
    funcName(arg1, argN);
*/
```



Recursive Function ?



```
// Declare
/*
 * dataType funcName(){
 *     // code to be executed
 *     if(condition){
 *         funcName();
 *     }
 *     return (value must match data type)
 * }
 */
void funcName(dataType par1, dataType parN){
    // code to be executed
    if(condition){
        funcName(arg1, argN);
    }
    return (value must match data type)
}
*/
```

```
// Call
/*
 * funcName();
 */
/*
 * funcName(arg1, argN);
*/
*/
```

Variable Scope (Local, Global & Block) ?

```
1 #include <iostream>
2
3 // global variable
4 ~/* dataType varName;
5   dataType varName = varVal;
6 */
7 */
8
```

```
9 int main()
10 {
11   // Local Variable
12   /* dataType varName;
13     dataType varName = varVal;
14   */
15
16   {
17     // Block Variable
18     /* dataType varName;
19       dataType varName = varVal;
20     */
21
22   }
23
24 }
25
26 }
27
```



Live Streaming

ENDING...



Thank you

#KEEPLEARNING
#KEEPSPIRITS

