

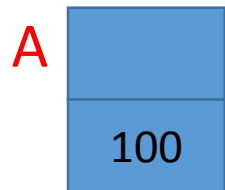
KONSEP LINKED LIST

KONSEP POINTER DAN LINKED LIST

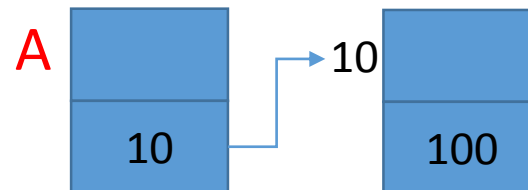
1. Dalam pengolahan data yang banyaknya tidak bisa ditentukan sebelumnya, maka diberikan satu fasilitas yang memungkinkan untuk menggunakan suatu perubah dinamis atau disebut variabel dinamis
2. Variabel dinamis merupakan suatu variabel yang akan dialokasikan pada saat dibutuhkan saja yaitu setelah program dieksekusi.

PERUBAH STATIS DAN DINAMIS

1. Pada perubah statis, isi Memory pada lokasi tertentu (nilai perubah) adalah data sesungguhnya yang akan diolah.
2. Pada perubah dinamis, nilai perubah adalah alamat lokasi lain yang menyimpan data sesungguhnya. Dengan demikian data yang sesungguhnya dapat dimasukkan secara langsung.



Statis



Dinamis

DEKLARASI POINTER

1. Pointer digunakan sebagai penunjuk ke suatu alamat memori.
2. Dalam pemrograman C++, Type Data Pointer dideklarasikan dengan bentuk umum :
`Type Data * Nama Variabel;`
3. Type Data dapat berupa sembarang type data, misalnya `char`, `int` atau `float`.
4. Sedangkan Nama variabel merupakan nama variabel pointer.

DEKLARASI POINTER

Contoh: (C++)

```
Void main()  
{  
  int x,y,*z;  
  x = 75;           //nilai x = 75  
  y = x;            //nilai y diambil dari nilai x  
  z = &x;           //nilai z menunjuk ke alamat pointer dari nilai x  
  getch();  
}
```

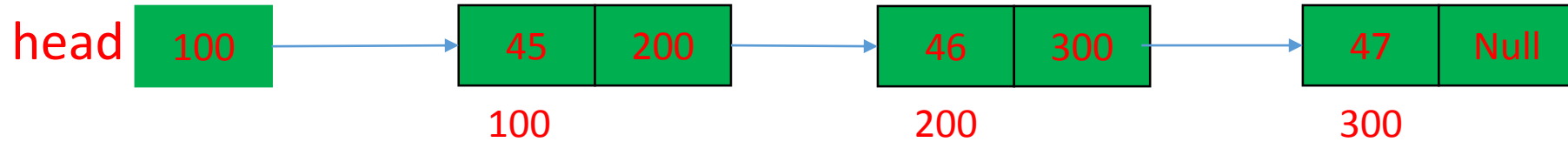
LINKED LIST

1. **Linked list** merupakan salah satu struktur data dinamis yang paling sederhana.
2. **Linked list** merupakan kumpulan komponen yang dinamakan nodes.
3. Setiap node terkecuali node terakhir berisi alamat node berikutnya.
4. Setiap node terdiri dari 2 komponen, yaitu:
 - a. Data → menyimpan informasi yang relevan
 - b. Link → menyimpan alamat dari node berikutnya

Node



LINKED LIST



→ Head atau first merupakan Alamat node pertama dalam daftar disimpan di lokasi terpisah.

→ Null mengindikasikan akhir dari list.

LINKED LIST

- Setiap node pada linked list terdapat dua komponen sehingga perlu dideklarasikan setiap node sebagai `class` atau `struct`.
- Tipe data pada setiap node bergantung pada aplikasi spesifik yang artinya jenis data apa yang sedang diproses.
- Sedangkan komponen link dari setiap node adalah pointer.
- Tipe data dari variabel pointer ini adalah tipe node itu sendiri.
- Contoh (dengan tipe data `int`)

```
struct nodeType
{
    int info;
    nodeType *link;
};
```

Deklarasi Variabel :

```
nodeType *head;
```


PERBEDAAN ARRAY DAN LINKED LIST

ARRAY	LINKED LIST
Statis	Dinamis
Penghapusan/penambahan data terbatas	Penghapusan/penambahan data tidak terbatas
Penghapusan array tidak mungkin	Penghapusan linked list mudah
Random access	Sequential access

KELEBIHAN LINKED LIST

1. Linked list merupakan struktur data dinamis, mengalokasikan memori yang dibutuhkan saat program dijalankan.
2. Operasi penambahan (insertion) atau penghapusan (deletion) node pada linked list mudah dilakukan.
3. Struktur data linier seperti stack dan queue mudah dieksekusi dengan menggunakan linked list.
4. Linked list dapat mengurangi waktu akses dan dapat dijalankan secara real time tanpa mengakibatkan memori overhead.

KEKURANGAN LINKED LIST

1. Linked list cukup sulit untuk menggunakan lebih banyak memori dikarenakan pointer membutuhkan penyimpanan ruang ekstra.
2. Node pada linked list harus dibaca secara berurutan dari awal karena linked list merupakan **sequential access**.
3. Node disimpan secara kontinu, sangat meningkatkan waktu yang dibutuhkan untuk mengakses elemen individu pada list.
4. Kesulitan saat linked list harus membalikkan traversing. Misal, single linked list sulit untuk di navigasi mundur, pada double linked list lebih mudah dibaca, memori terbuang sia-sia mengalokasikan ruang untuk back pointer.

JENIS LINKED LIST

1. Single linked list
2. Double linked list
3. Circular linked list
4. Circular double linked list

SINGLE LINKED LIST

1. Single linked list merupakan semua node dihubungkan bersama-sama secara berurutan.
2. Hanya ada satu arah traversal yaitu **forward traversal**.
3. Single linked list terdiri atas dua bagian yaitu **data** dan **link**.
Data berisi elemen.
Link berisi alamat node selanjutnya.
4. Last node pada bagian link bernilai Null artinya akhir pada list.



SINGLE LINKED LIST

DOUBLE LINKED LIST

1. Double linked list merupakan semua node dihubungkan bersama-sama secara berurutan.
2. Ada dua arah traversal yaitu **forward traversal** dan **backward traversal**.
3. Double linked list terdiri atas tiga bagian yaitu satu **data** dan dua **link**.
Data berisi elemen.
First link berisi alamat node sebelumnya.
Second link berisi alamat node selanjutnya.
4. First node link dan last node link bernilai Null artinya akhir pada list.



DOUBLE LINKED LIST

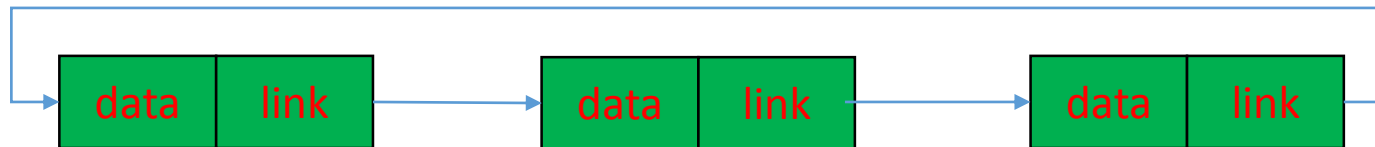
CIRCULAR LINKED LIST

1. Circular linked list merupakan semua node dihubungkan bersama-sama secara berurutan.
2. Hanya ada satu arah traversal dengan cara melingkar yaitu **forward traversal**.
3. Single linked list terdiri atas dua bagian yaitu **data** dan **link**.

Data berisi elemen.

Link berisi alamat node selanjutnya.

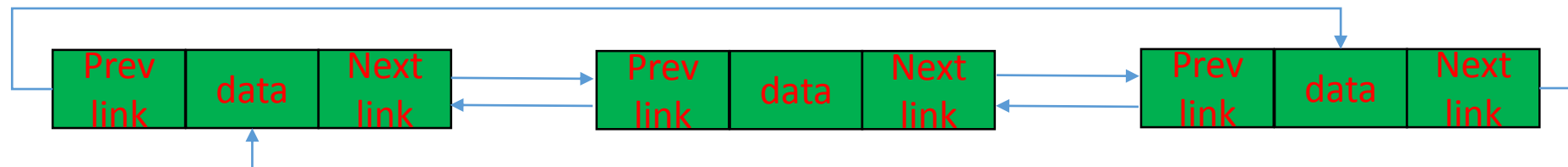
4. Last node pada bagian link berisi alamat dari first node.



CIRCULAR LINKED LIST

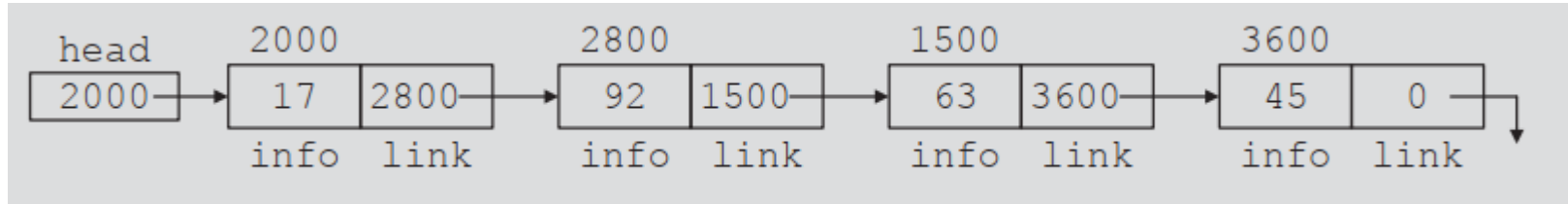
CIRCULAR DOUBLE LINKED LIST

1. Circular Double linked list merupakan semua node dihubungkan bersama-sama secara berurutan.
2. Ada dua arah traversal dengan cara melingkar yaitu **forward traversal** dan **backward traversal**.
3. Double linked list terdiri atas tiga bagian yaitu satu **data** dan dua **link**.
Data berisi elemen.
First link berisi alamat node sebelumnya.
Second link berisi alamat node selanjutnya.
4. First node pada Prev link berisi alamat last node dan last node pada Next link berisi alamat first node.



CIRCULAR DOUBLE LINKED LIST

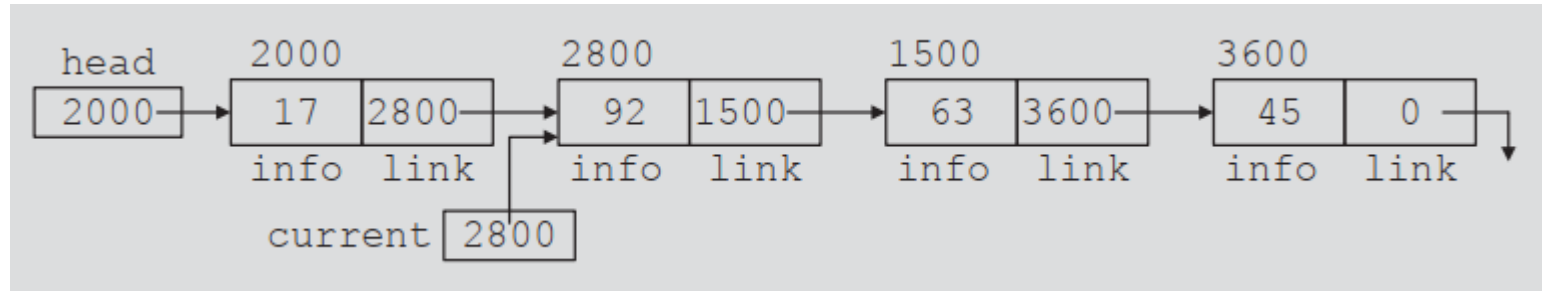
CONTOH LINKED LIST



PENJELASAN

	Value	Explanation
head	2000	
head->info	17	Because head is 2000 and the info of the node at location 2000 is 17
head->link	2800	
head->link->info	92	Because head->link is 2800 and the info of the node at location 2800 is 92

CONTOH LINKED LIST



PENJELASAN

	Value
current	2800
current->info	92
current->link	1500
current->link->info	63
head->link->link	1500
head->link->link->info	63
head->link->link->link	3600
current->link->link->link	0 (that is, NULL)
current->link->link->link->info	Does not exist (run-time error)

OPERASI PADA LINKED LIST

- Creation

Operasi Creation berfungsi untuk membuat linked list baru.

- Insertion

Operasi Insertion berfungsi untuk menambahkan node baru pada linked list. Penambahan node baru dapat diawal, diakhir atau pada posisi yang spesifik pada linked list. Apabila link list empty maka menjadi head atau first node.

- Deletion

Operasi Deletion berfungsi untuk menghapus node pada linked list. Penghapusan node dapat diawal, diakhir atau pada posisi yang spesifik pada linked list.

OPERASI PADA LINKED LIST

- Traversing

Proses Traversing merupakan proses melewati semua node pada linked list dari satu ujung ke ujung lainnya. Jika awal traversing mulai dari node awal ke node akhir disebut **forward traversing**. Jika awal traversing mulai dari node akhir ke node awal disebut **backward traversing**.

- Searching

Proses Searching merupakan proses mengakses node yang diinginkan pada list. Proses Searching dimulai dari node ke node dan membandingkan data pada node dengan key.

- Concatenation

Operasi concatenation merupakan proses menambahkan second list ke akhir dari first list. Penggabungan 2 list dapat dilakukan sehingga list yang dihasilkan menjadi lebih besar.

- Display

Operasi display digunakan untuk mencetak informasi setiap node.

CONTOH OPERASI PADA LINKED LIST

TRAVERSING

Contoh kode traversal:

```
current = head;
while (current != NULL)
{
    //Process current
    current = current->link;
}
```

Misal: head pada linked list berupa angka maka kode berikut ini menampilkan data yang disimpan pada setiap node:

```
current = head;
while (current != NULL)
{
    cout << current->info << " ";
    current = current->link;
}
```

CONTOH OPERASI PADA LINKED LIST

INSERTION DAN DELETION

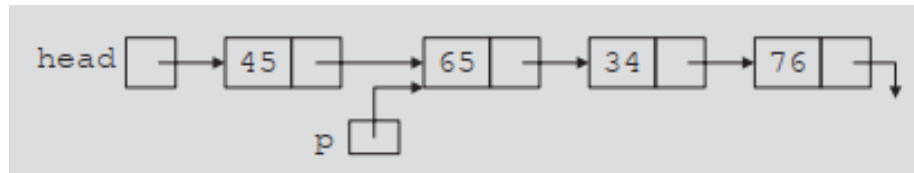
Misal, Definisi node adalah sebagai berikut (type data (info)= `int`):

```
struct nodeType
{
    int info;
    nodeType *link;
};
```

Deklarasi variabel:

```
nodeType *head, *p, *q, *newNode;
```

Linked list before insertion:



CONTOH OPERASI PADA LINKED LIST

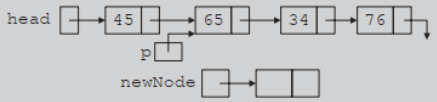
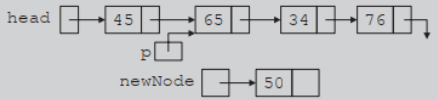
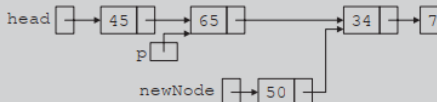
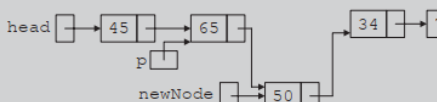
INSERTION

Misalkan p menunjuk ke node dengan info 65, dan node baru dengan info 50 dibuat dan disisipkan setelah p.

Perhatikan pernyataan berikut:

```
newNode = new nodeType;  
newNode->info = 50;           //create newNode  
newNode->link = p->link;      //store 50 in the new node  
p->link = newNode;
```

Insertion linked list:

Statement	Effect
<code>newNode = new nodeType;</code>	
<code>newNode->info = 50;</code>	
<code>newNode->link = p->link;</code>	
<code>p->link = newNode;</code>	

CONTOH OPERASI PADA LINKED LIST

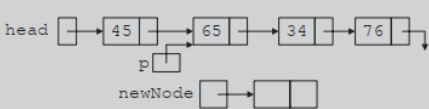
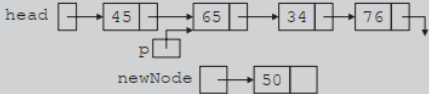
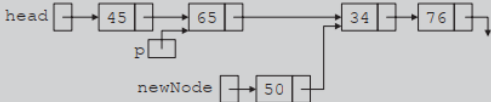
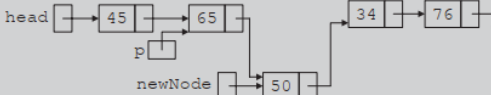
INSERTION

Misalkan p menunjuk ke node dengan info 65, dan node baru dengan info 50 dibuat dan disisipkan setelah p.

Perhatikan pernyataan berikut:

```
newNode = new nodeType;           //create newNode
newNode->info = 50;                 //store 50 in the new node
newNode->link = p->link;
p->link = newNode;
```

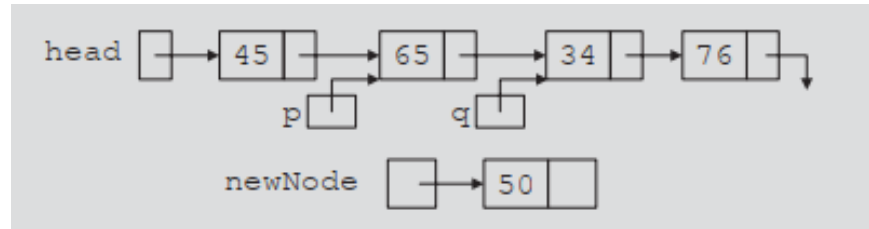
Insertion linked list:

Statement	Effect
<code>newNode = new nodeType;</code>	
<code>newNode->info = 50;</code>	
<code>newNode->link = p->link;</code>	
<code>p->link = newNode;</code>	

CONTOH OPERASI PADA LINKED LIST

INSERTION

Insertion linked list menggunakan 2 pointer:



Insert statement newNode antara p dan q:

```
newNode->link = q;
```

```
p->link = newNode;
```

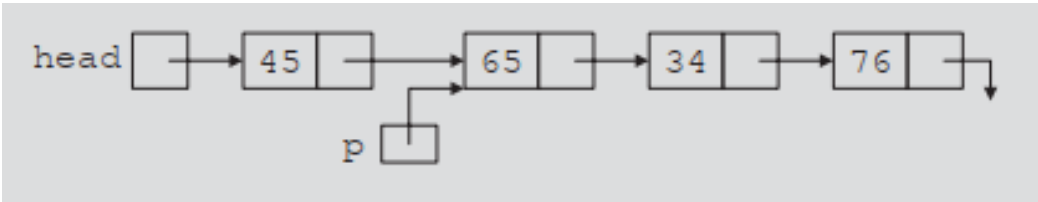
Insertion linked list dengan dua pointer:

Statement	Effect
<code>p->link = newNode;</code>	<p>The diagram shows the linked list after the statement <code>p->link = newNode;</code>. The 'newNode' (50) is now pointed to by 'p'. The original link of 'p' (which was 'q') is now lost, and 'q' still points to the node with value 65.</p>
<code>newNode->link = q;</code>	<p>The diagram shows the linked list after the statement <code>newNode->link = q;</code>. The 'newNode' (50) now points to the node with value 65 (which 'q' points to). The original link of 'p' is restored.</p>

CONTOH OPERASI PADA LINKED LIST

DELETION

Linked list before deletion (sebagai contoh akan dilakukan penghapusan pada info 34):



Program untuk mendelete info 34:

```
q = p->link;
p->link = q->link;
delete q;
```

Statement	Effect
<code>q = p->link;</code>	
<code>p->link = q->link;</code>	
<code>delete q;</code>	

PEMBELAJARAN MANDIRI

- Pelajari pada buku dan sumber-sumber informasi lainnya tentang contoh program pada masing-masing operasi di Linked list. Serta bagaimana perbedaan program apabila operasi-operasi tersebut dilakukan di awal, tengah dan akhir pada Node.

*****SELAMAT BELAJAR*****

TERIMA KASIH