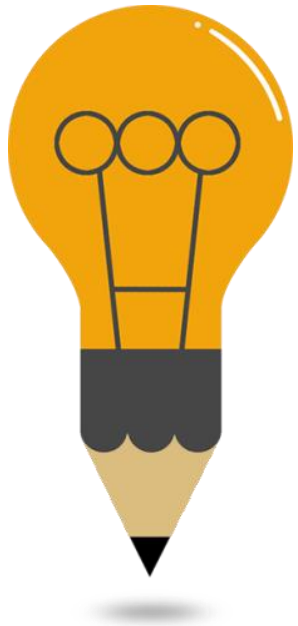
STRUKTUR DATA

Pertemuan 5

Ratih Ngestrini, Nori Wilantika

Agenda Pertemuan



1. Operasi-Operasi Single Linked List
2. Operasi-Operasi Double Linked List

OPERASI-OPERASI SINGLE LINKED LIST

Operasi Pada Linked List

1. **Menambahkan node (insert)**
   - Insert sebagai node awal (head) dari linked list
   - Insert sebagai node akhir (tail) dari linked list
   - Insert setelah node tertentu
   - Insert sebelum node tertentu
2. **Menghapus node (delete)**
   - Delete node pertama (head) dari linked list
   - Delete node terakhir (tail)
   - Delete pada node tertentu
3. **Penelusuran (Traversal)**

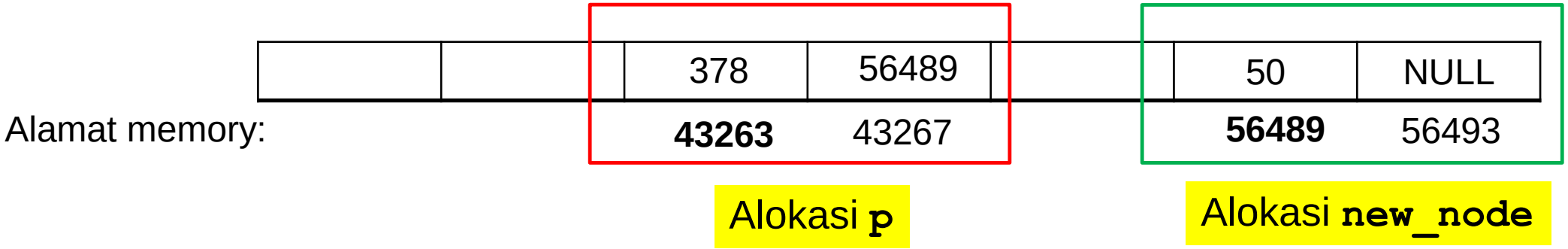
Ingat!!

Untuk mengakses array, kita memakai nama variabel array dan indexnya

Untuk mengakses linked list (node-node di dalamnya) yang diketahui adalah **node/pointer head**
(karena dari head kita bisa baca seluruh elemen dalam linked list)

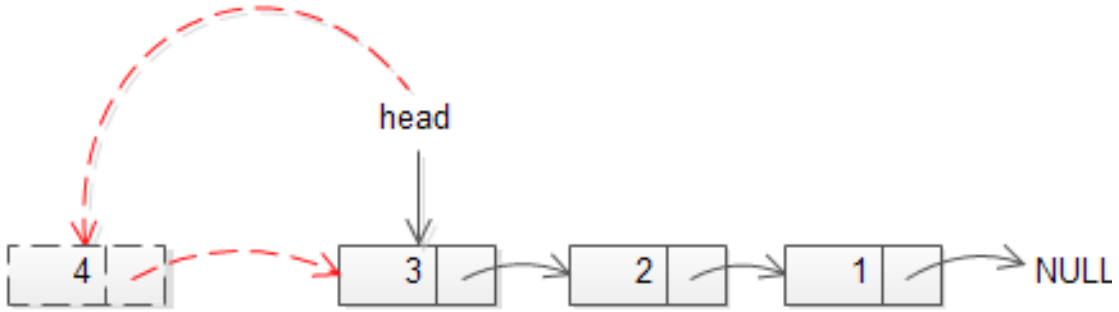
Apa yang terjadi jika kita menghubungkan 2 node (`p` dan `new_node`)?

```
new_node->data = 50;
new_node->next = NULL;

p->next = new_node ;
```



Insert sebagai node awal (*head*) dari linked list

Contoh: Insert node dengan data = 4 sebagai head linked list 3->2->1 sehingga menjadi 4->3->2->1

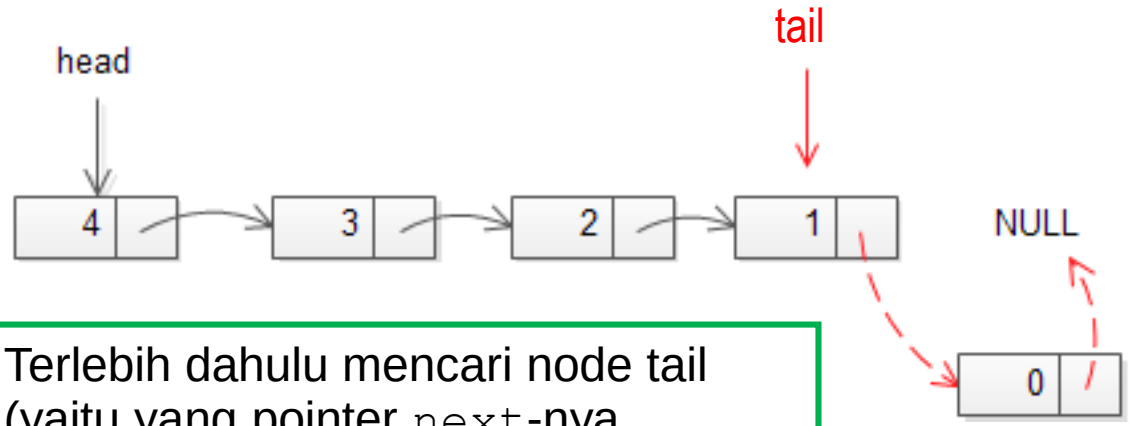



```
mynode insert_head(mynode head, int nilai){
    mynode new_node = createNode(nilai);
    new_node->next = head;
    head = new_node;

    return(head);
}
```

- Membuat node baru `new_node`
- Mengarahkan pointer `next` dalam `new_node` ke `head`, sehingga head yang baru adalah `new_node`
- Karena linked list sudah berubah, maka return-kan head yang baru

Insert sebagai node akhir (*tail*) dari linked list (Append)

Contoh: Insert node dengan data = 0 sebagai tail linked list **3->2->1** sehingga menjadi **3->2->1->0**

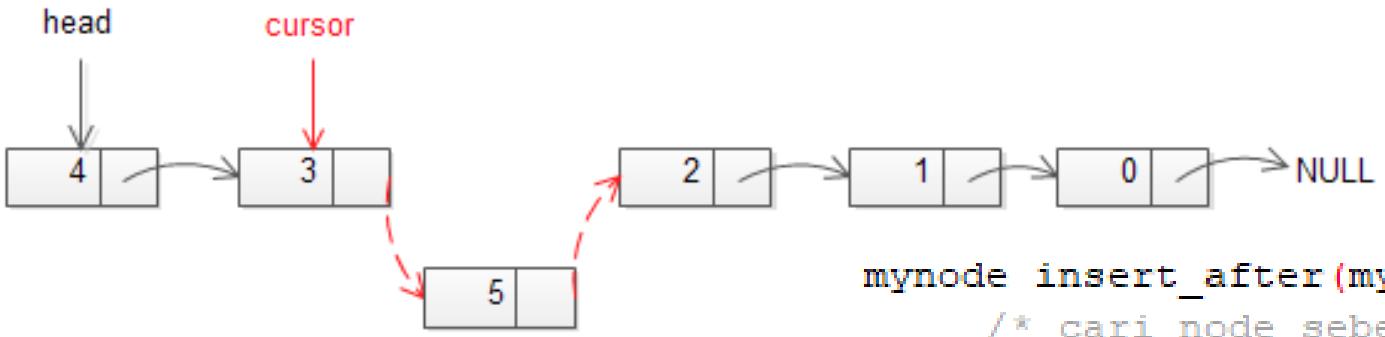


- Terlebih dahulu mencari node tail (yaitu yang pointer `next`-nya mengarah ke NULL) dengan melakukan iterasi dari head.
- Setelah ditemukan node tail, buat node baru `new_node`.
- Mengarahkan pointer `next` dari tail sebelumnya ke `new_node`, sehingga `new_node` menjadi tail.

```
mynode insert_tail(mynode head, int nilai){
    /* iterasi mencari node terakhir*/
    mynode tail = head;
    while(tail->next != NULL)
        tail = tail->next;

    /* buat node baru */
    mynode new_node = createNode(nilai);
    tail->next = new_node;

    return(head);
}
```

Insert setelah node tertentu (misal node dengan nilai tertentu)

Contoh: Insert node dengan data = 5 setelah node yang ditandai dengan "**`cursor`**" (data = 3)



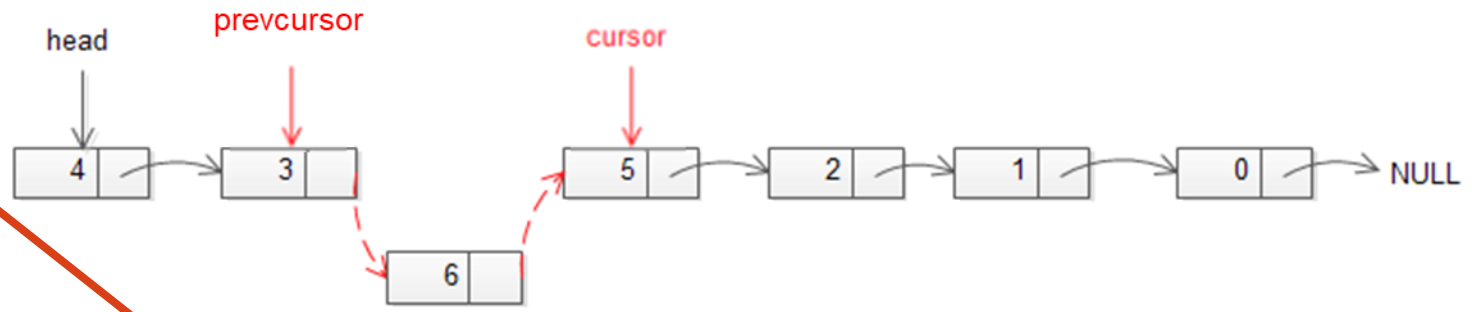
- Terlebih dahulu mencari node cursor (yaitu node yang mempunyai nilai `prev_nilai`) dengan iterasi dari node head.
- Jika sudah ditemukan, buat node baru `new_node`. Arahkan pointer next `new_node` ke alamat yang ditunjuk pointer next cursor. Dan arahkan pointer next cursor ke new_node.

```
mynode insert_after(mynode head, int nilai, int prev_nilai){
     /* cari node sebelumnya, starting from the first node*/
    mynode cursor = head;
    while(cursor->value != prev_nilai)
        cursor = cursor->next;

    mynode new_node = createNode(nilai);
    new_node->next = cursor->next;
    cursor->next = new_node;

    return(head);
}
```

Insert sebelum node tertentu (misal node dengan nilai tertentu)

Contoh: Insert node dengan data = 6 sebelum node yang ditandai dengan "**`cursor`**" (data = 5)

```
mynode insert_before(mynode head, int nilai, int next_nilai){
    if (head->value == next_nilai)
        head = insert_head(head, nilai);
    else
    {
        mynode cursor, prevcursor;
        cursor = head;
        do
        {   prevcursor = cursor;
            cursor = cursor->next;
        }
        while (cursor->value != next_nilai);

        mynode new_node = createNode(nilai);
        new_node->next = cursor;
        prevcursor->next = new_node;
    }

    return(head);
}
```

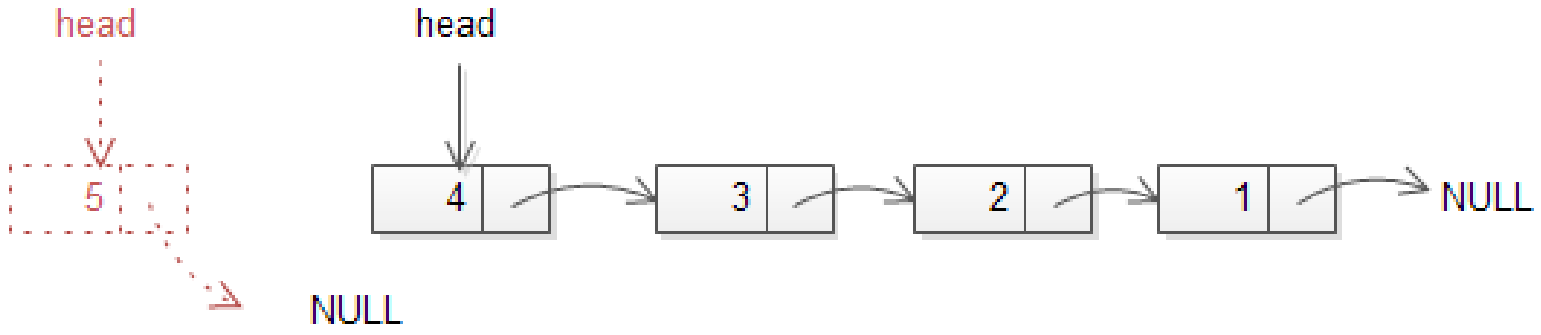


If: Jika data yang dicari berada pada awal Linked List atau head berarti gunakan fungsi sebelumnya

Else: Jika data yang dicari tidak berada pada awal Linked List

Menggunakan node bantuan `prevcursor` untuk menyimpan node sebelumnya, agar bisa dihubungkan dengan node baru

Delete node pertama (head) dari linked list




```
mynode remove_first(mynode head){
    if(head == NULL)
        return;

    mynode first = head;
    head = head->next;
    first->next = NULL;

    free(first);

    return(head);
}
```

Jika linked list empty (head == null) maka keluar dari fungsi.

Jika tidak:

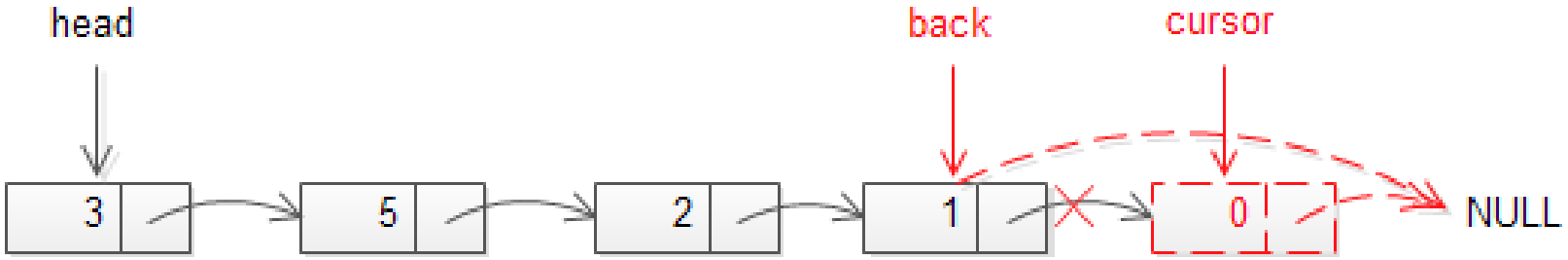
- Node first diarahkan pada node head
- Node head diarahkan pada node setelah head
- Bebaskan node first (secara otomatis data pada node pertama terhapus)

Fungsi `free()` pada Linked List

- Membebaskan memory yang dialokasi untuk node tersebut

```
void free_node(mynode node){
     free(node);
}
```

Delete node terakhir (tail)



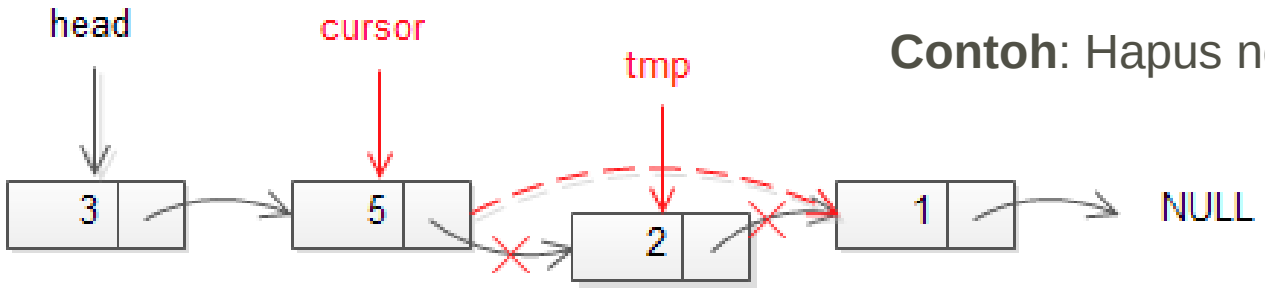

```
mynode remove_last(mynode head){
    if(head == NULL)
        return;

    mynode cursor = head;
    mynode back = NULL;
    while(cursor->next != NULL)
    {
        back = cursor;
        cursor = cursor->next;
    }
    if(back != NULL)
        back->next = NULL;

    free(cursor);

    return(head);
}
```

Jika linked list empty keluar dari fungsi.

Jika tidak:

- Iterasi dari head untuk mencari node terakhir (cursor) dan node sebelum terakhir (back)
- Setelah ditemukan, arahkan pointer next dari node back ke NULL dan bebaskan node terakhir (cursor).

Delete pada node tertentu (node dengan nilai tertentu)



Contoh: Hapus node dengan data = 2

Iterasi dari node head ke node terakhir (menggunakan cursor).

Sebelum *free*/membebaskan memory node yang ditunjuk cursor, buat dulu temporary node **tmp** untuk menyimpan alamat node selanjutnya (karena jika node yang ditunjuk cursor sudah di-*free*-kan, alamat next-nya sudah tidak ada)

```
mynode remove_middle(mynode head, int nilai){
    mynode cursor = head;
    while(cursor != NULL)
    {
        if(cursor->next->value == nilai)
            break; //keluar dari iterasi
        cursor = cursor->next;
    }

    if(cursor != NULL)
    {
        mynode tmp = cursor->next;
        cursor->next = tmp->next;
        tmp->next = NULL;
        free(tmp);
    }

    return(head);
}
```

Delete Linked List (Dispose)

- Penting untuk menghapus seluruh memory yang digunakan node-node pada linked list ketika sudah tidak digunakan/diperlukan

```
mynode dispose(mynode head)
{
    mynode cursor, tmp;
    if(head != NULL)
    {
        cursor = head;
        while(cursor != NULL)
        {
            tmp = cursor->next;
            free(cursor);
            cursor = tmp;
        }
    }
    head = NULL;

    return(head);
}
```

- Satu per satu menghapus node dari head ke terakhir. Iterasi dari node head ke node terakhir (menggunakan cursor).
- Sebelum *free*/membebaskan memory node yang ditunjuk cursor, buat dulu temporary node tmp untuk menyimpan alamat node selanjutnya (karena jika node yang ditunjuk cursor sudah di-*free*-kan, alamat next-nya sudah tidak ada).
- Setelah selesai iterasi, free-kan node head karena cursor dimulai dari head->next jadi head belum terbebaskan.

Ringkasan

So far, kita sudah mempunyai fungsi-fungsi:

- `createNode()`
- `insert_head()`
- `insert_tail()`
- `insert_after()`
- `insert_before()`
- `remove_first()`
- `remove_last()`
- `remove_middle()`
- `dispose()`

Fungsi-fungsi di atas bukan satu-satunya solusi, **algoritma bisa berbeda-beda** tetapi fungsi/tujuannya sama.

Implementasi di program C (coding)

- **Jangan lupa**: misalnya jika kita ingin memakai **void()** akan tetapi kita ingin mengubah nilai aslinya maka harus ***pass by reference*** (buka kembali slide-slide sebelumnya)
- Contoh: fungsi dalam `insert_head` harus diubah menjadi:

```
mynode insert_head(mynode head, int nilai){
    mynode new_node = createNode(nilai);
    new_node->next = head;
    head = new_node;

    return(head);
}
```

```
void insert_head(mynode *head, int nilai){
    mynode new_node = createNode(nilai);
    new_node->next = *head;
    *head = new_node;
}
int main(){
    mynode head = NULL;
    mynode dua = NULL;
    head = (mynode)malloc(sizeof(struct node));
    dua = (mynode)malloc(sizeof(struct node));

    head->value = 10;
    head->next = dua;
    dua->value = 20;
    dua->next = tiga;

    insert_head(&head, 99);

    return 0;
}
```

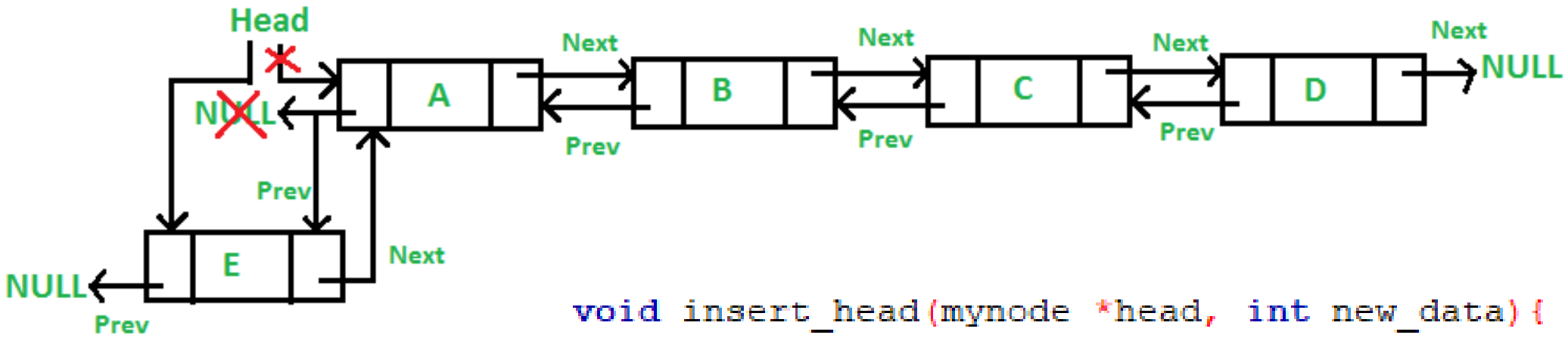
Bedakan jika fungsi yang kita buat bukan return value tapi void()!

OPERASI-OPERASI
DOUBLE LINKED LIST

Operasi pada Double Linked List

1. **Menambahkan node (insert)**
   - Insert sebagai node awal (head)
   - Insert sebagai node akhir (tail)
   - Insert setelah node tertentu
   - Insert sebelum node tertentu
2. **Menghapus node (delete)**
   - Delete node pertama (head)
   - Delete node terakhir (tail)
   - Delete pada node tertentu
3. **Penelusuran (Traversal)**

Insert sebagai node awal (head)

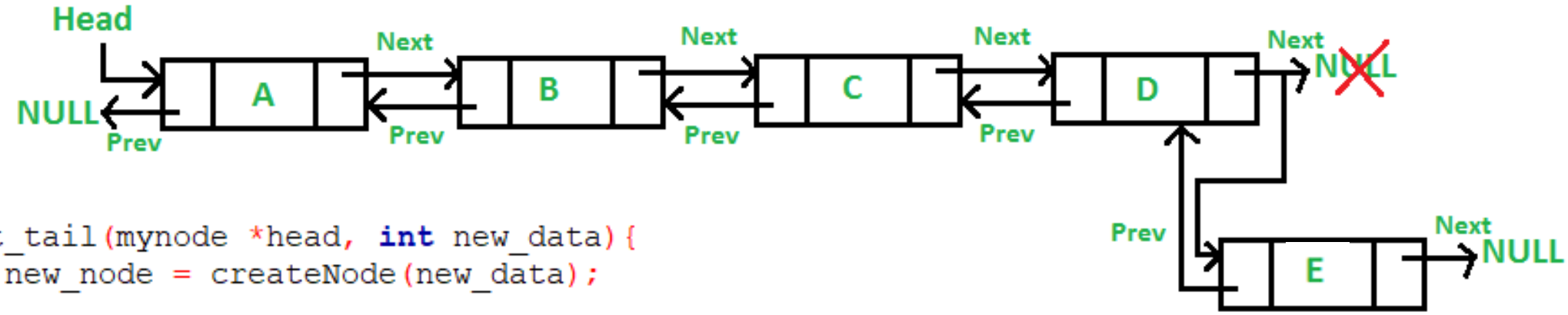



```
void insert_head(mynode *head, int new_data){
    mynode new_node = createNode(new_data);
    new_node->next = *head;

    if(*head != NULL)
        (*head)->prev = new_node;

    *head = new_node;
}
```

Jika DLL tidak kosong, hubungkan dengan node baru

Ingat: `mynode *head` sama saja dengan `struct node **head` → double pointer (menyimpan alamat memory pointer)

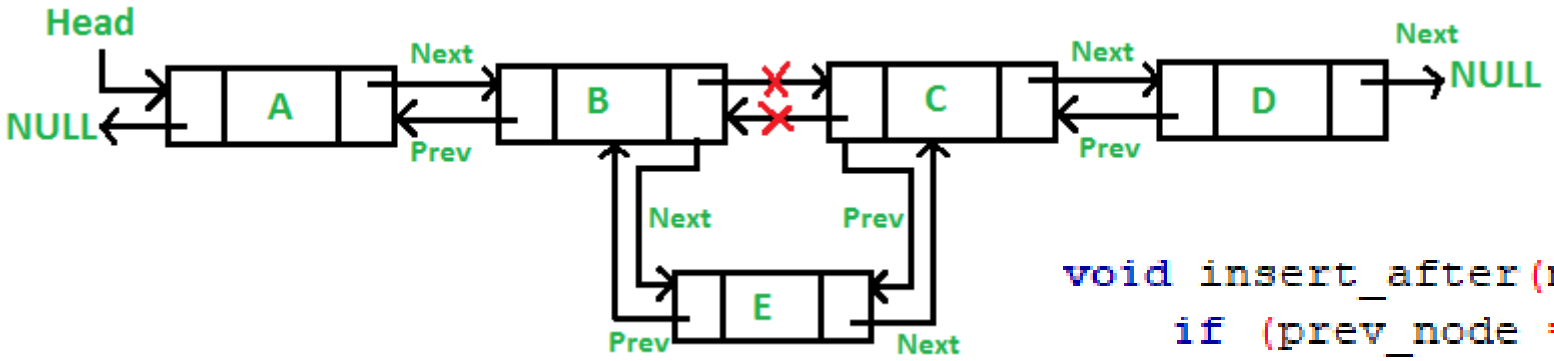
Insert sebagai node akhir (tail)




```
void insert_tail(mynode *head, int new_data){
    mynode new_node = createNode(new_data);

    if(*head == NULL){
        *head = new_node;
        return;
    }

    mynode cursor = *head;
    while(cursor->next != NULL)
        cursor = cursor->next;

    cursor->next = new_node;
    new_node->prev = cursor;
}
```

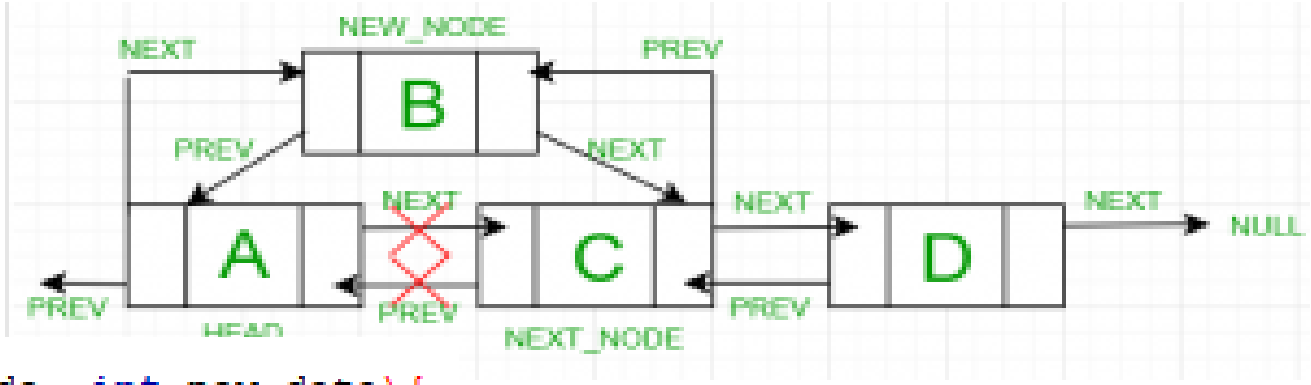
Mencari node tail dari head

Insert setelah node tertentu




```
void insert_after(mynode prev_node, int new_data){
    if (prev_node == NULL) {
        printf("Previous Node tidak boleh NULL");
        return;
    }

    mynode new_node = createNode(new_data);
    new_node->next = prev_node->next;
    prev_node->next = new_node;
    new_node->prev = prev_node;

    if (new_node->next != NULL)
        new_node->next->prev = new_node;
}
```

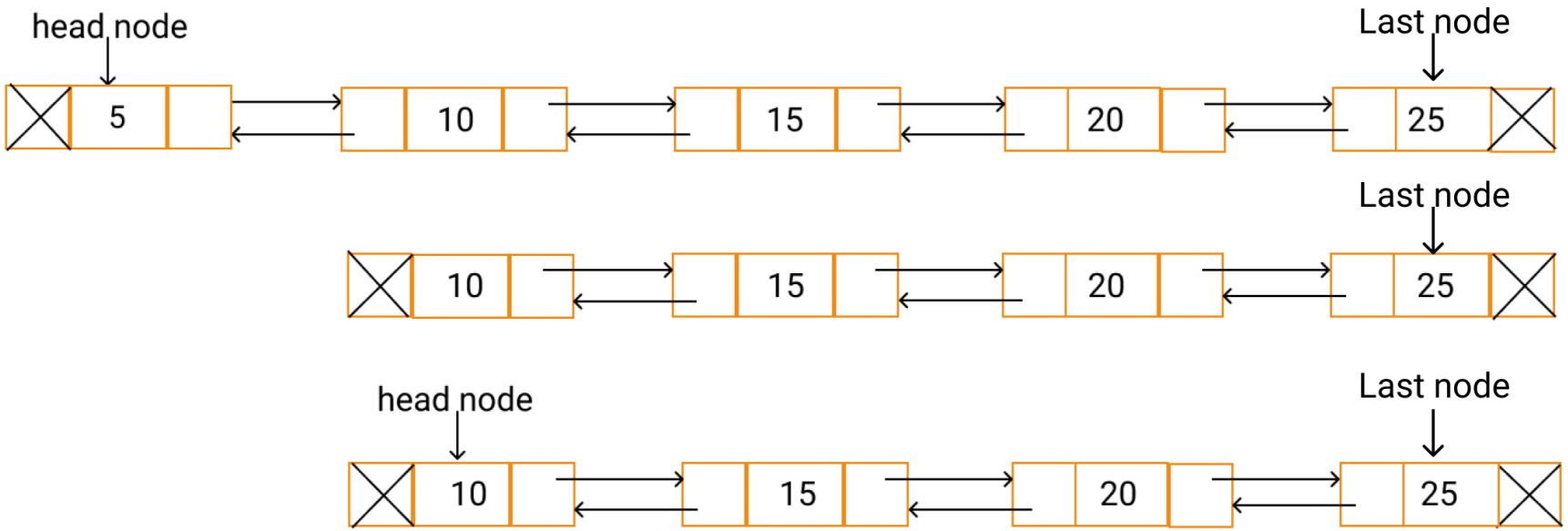
Insert sebelum node tertentu




```
void insert_before(mynode next_node, int new_data){
    if (next_node == NULL) {
        printf("Next Node tidak boleh NULL");
        return;
    }

    mynode new_node = createNode(new_data);
    new_node->prev = next_node->prev;
    next_node->prev = new_node;
    new_node->next = next_node;

    if (new_node->prev != NULL)
        new_node->prev->next = new_node;
}
```

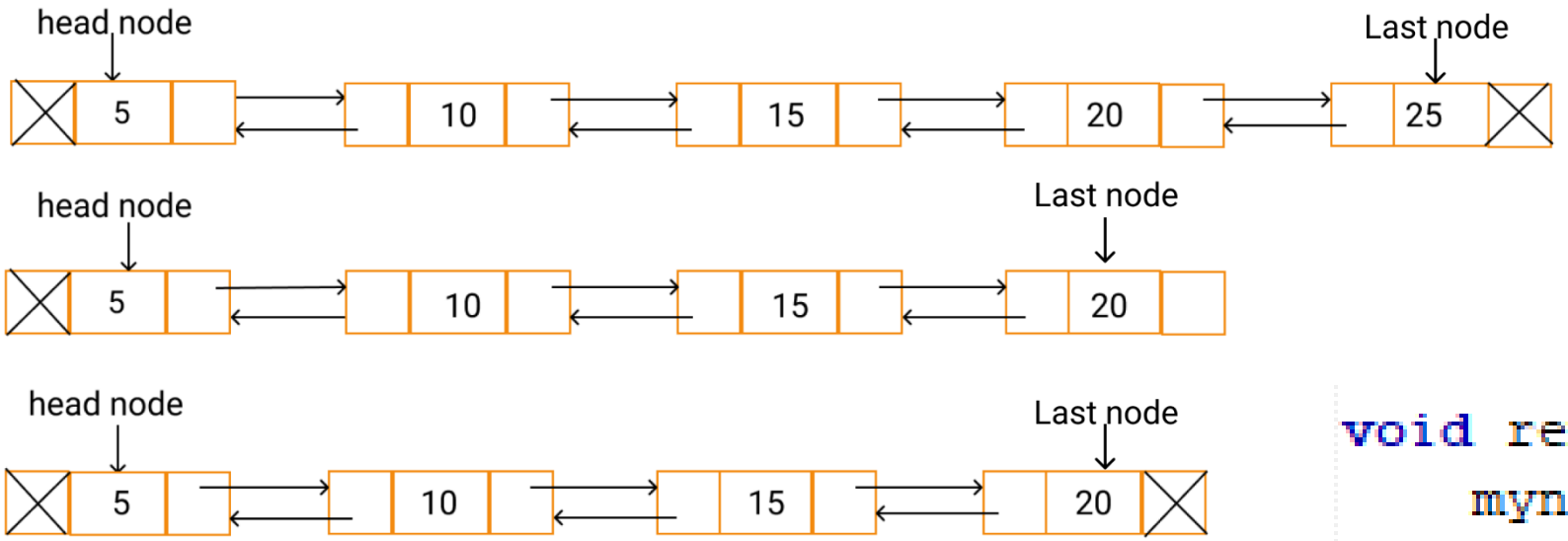
Ubah pointer Next dari previous node ke node baru

Delete node pertama (head)



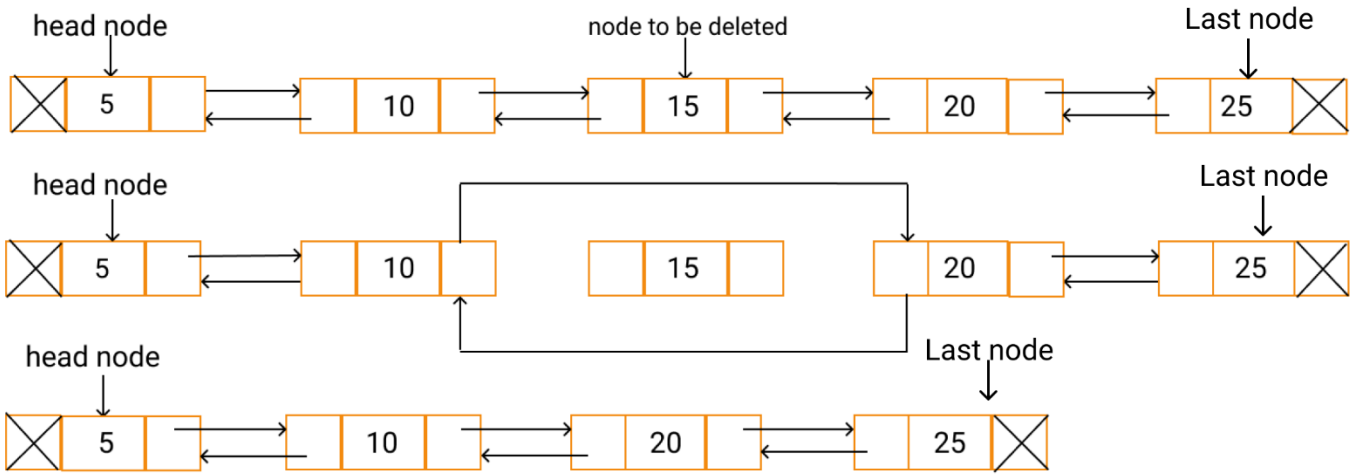

```
void remove_first(mynode *head){
     mynode temp = *head;
     *head = (*head)->next;
     (*head)->prev = NULL;

     free(temp);
}
```

Delete node terakhir (tail)




```
void remove_end(mynode tail){
    mynode temp = tail;
    tail = tail->prev;

    if(tail != NULL)
        tail->next = NULL;

    free(temp);
}
```

Delete pada node tertentu



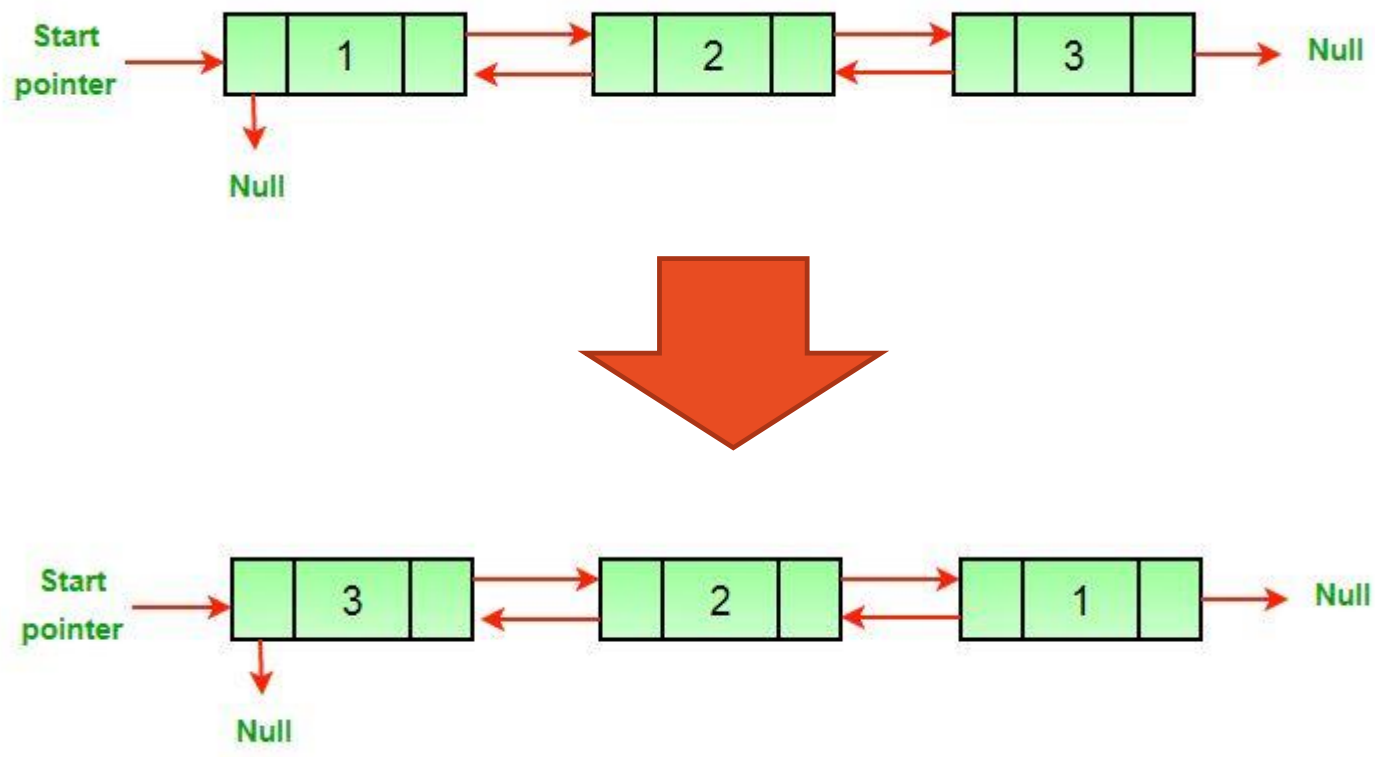

```
void remove_middle(mynode head, int position){
    mynode temp = head;
    for(int i=1; i<position && temp!=NULL; i++)
    {
        temp = temp->next;
    }

    if(temp != NULL)
    {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;

        free(temp);
    }
    else
    {
        printf("Posisi tidak valid!\n");
    }
}
```

Aplikasi Linked List di Dunia Nyata

- Image viewer
- Previous & next page di web browser
- Playlist di music player

Latihan: Membalik Double Linked List

- Buatlah program untuk membalik nilai-nilai dalam double linked list (tail ke head), seperti yang diilustrasikan oleh gambar berikut.



TERIMA KASIH