

# Linked List (3)

Dr. Anto Satriyo Nugroho, M.Eng

Email: [asnugroho@gmail.com](mailto:asnugroho@gmail.com)

Web: <http://asnugroho.net/lecture/ds.html>

# Outline

- Linked List
  - Bagaimana membuat STACK dengan linked-list ? (50 minutes)
  - Bagaimana membuat QUEUE dengan linked –list ? (PR)
- Circular List & Bidirectional List

## PR (29 November 2008)

- Rancanglah bagaimana membuat program stack, tetapi tidak memakai array, melainkan memakai linked-list (Dengan kata lain: Selesaikan fungsi push, pop dan stack\_print pada linkedliststack.c)
- Hint:  
Penambahan dan penghapusan elemen pada stack harus bisa dijalankan pada  $O(1)$ . Karena itu penambahan dan penghapusan data harus dilakukan pada sel yang terdepan

# Fungsi PUSH

```
void push(int val)
{
    cell *p;
    // memakai fungsi malloc untuk membuat sel baru
    if((p=(cell*)malloc(sizeof(cell)))==NULL) {
        fprintf(stderr,"can not allocate memory\n");
        exit(1);
    }
    // Memasukkan data val ke sel, dan menyesuaikan koneksi antar sel
    // lewat pointer
    else {
        p->value=val;
        p->next=header;
        header=p;
    }
}
```

# Fungsi POP

```
void pop()
{
    cell *p;
    // memeriksa apakah stack dalam kondisi kosong atau tidak
    if(header==NULL) {
        fprintf(stderr,"Stack underflow\n");
    }
    // mengubah koneksi antar sel lewat pointer, dan membebaskan
    // memori sel yang akan dihapus dengan fungsi free()
    else {
        p=header;
        header=p->next;
        free(p);
    }
}
```

# Fungsi stack\_print

```
void stack_print()
{
    cell *p;
    printf("¥tStack :¥t");

    // posisi p digerakkan mulai dari header hingga NULL,
    // sambil menampilkan value tiap sel
    for(p=header;p!=NULL;p=p->next)
        printf("%3d ",p->value);

    printf("¥n");
}
```

# Kelebihan & Kelemahan

## Kelebihan

1. Penambahan dan penghapusan data dapat dilakukan dengan cepat, yaitu  $O(1)$
2. Selama memory masih tersedia, penambahan data bisa terus dilakukan. Dengan demikian tidak ada kekhawatiran terjadinya stack overflow.

## Kelemahan

1. Setiap sel tidak hanya menyimpan value saja, melainkan juga pointer ke sel berikutnya. Hal ini menyebabkan implementasi stack memakai linked list akan memerlukan memory yang lebih banyak daripada kalau diimplementasikan dengan array.
2. Tiap elemen pada linked list hanya bisa diakses dengan cara sekuensial, sehingga lambat, yaitu  $O(n)$ .

## PR (1 Desember 2008)

- Rancanglah bagaimana membuat program QUEUE, tetapi tidak memakai array, melainkan memakai linked-list
- Hint:  
Penambahan dan penghapusan elemen pada QUEUE harus bisa dijalankan pada  $O(1)$ .

# Beberapa Jenis Struktur Data

## 1. Array

1. Linear List
2. Stack
3. Queue

## 2. List

1. Linked List
2. Circular List
3. Bidirectional List
4. Multi list structure

## 3. Tree Structure

# Circular List

## Apakah Circular List itu ?

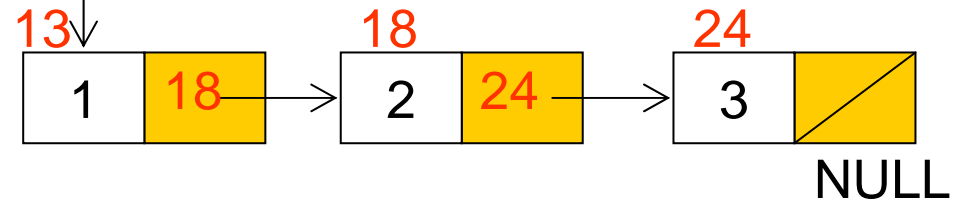
- Semua sel dalam list disambungkan dengan pointer, dan sel yang terakhir disambungkan dengan sel pertama
- Struktur data berbentuk cincin
- Tidak ada sel pertama maupun sel terakhir dalam list, karena setiap sel disambungkan oleh pointer sehingga berbentuk cincin.

# Linked List vs Circular List

Variabel ptr



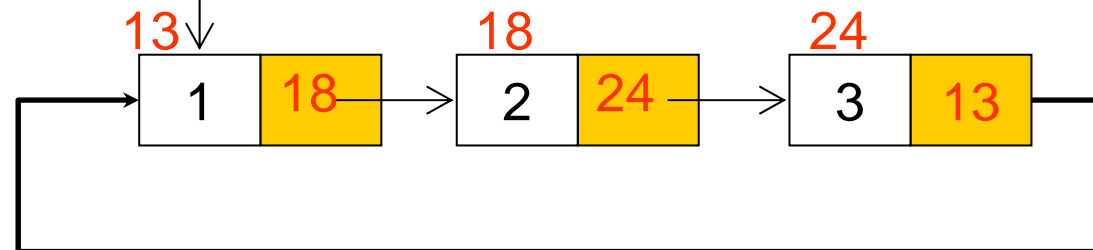
Linked list



Variabel ptr



Circular list

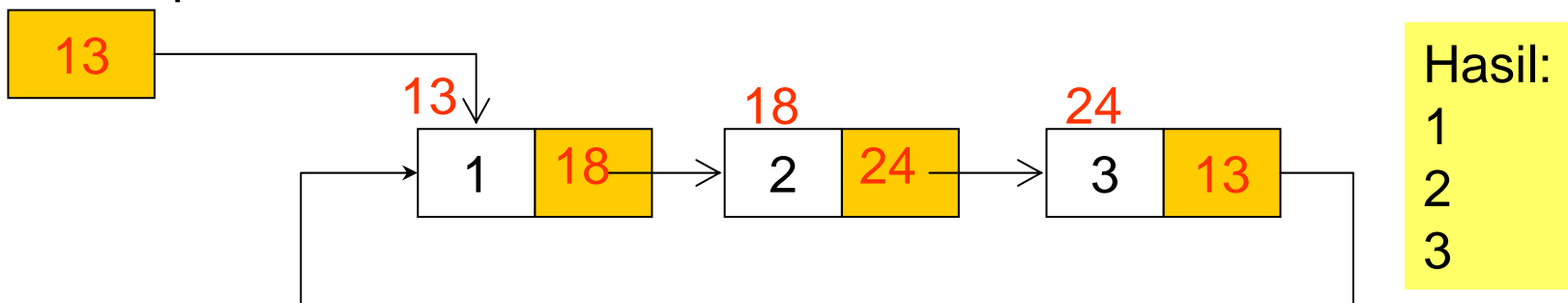


Kembali ke sel terdepan

# Cara akses tiap sel pada Circular List

```
struct CELL *ptr, *p;  
if(ptr != NULL) {  
    p = ptr;  
    do {  
        printf("%d\n",p->value)  
        /* tampilkan value pada sel yang ditunjuk oleh p */  
        p = p->next;  
    } while (p != ptr);  
    // Setelah semua bagian dalam loop dieksekusi,  
    // evaluasilah apakah p!= ptr  
}
```

Variabel ptr



## Cara ini tidak benar !

```
struct CELL *ptr, *p;
if(ptr != NULL) {
    p = ptr;
    while (p != ptr) {
        printf("%d\n",p->value)
        p = p->next;
    }
}
```

Cara ini tidak benar. Bagian dalam loop tidak akan dijalankan karena sebelum masuk loop syarat `p!=ptr` tidak akan pernah terpenuhi

# Circular List memakai head

```
struct CELL *ptr, *p;
```

```
if(ptr != NULL) {
```

```
  p = ptr;
```

```
  do {
```

```
    printf("%d\n", p->value
```

```
    p = p->next;
```

```
  } while (p != ptr);
```

```
}
```

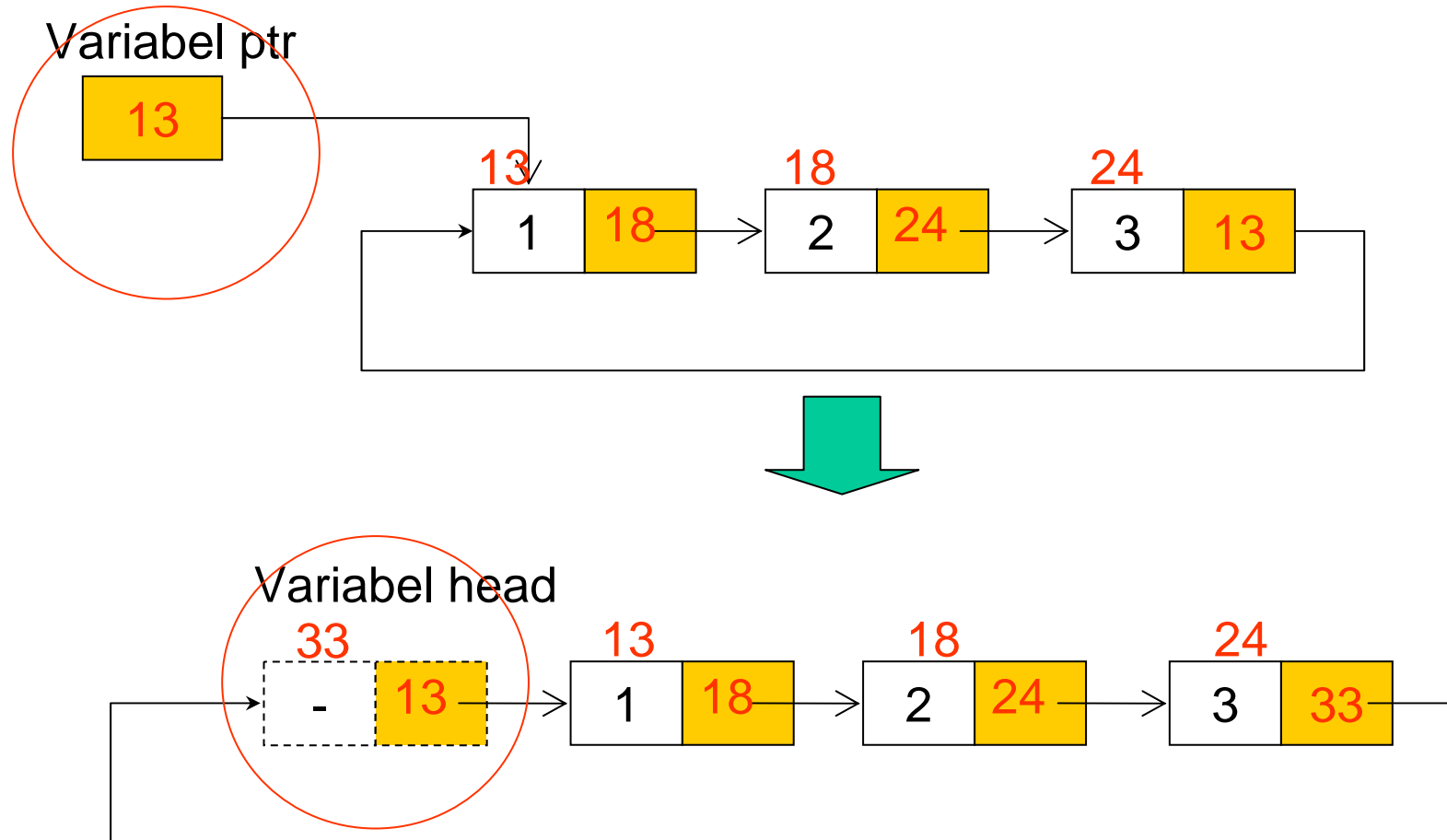
Saat ptr==NULL  
berarti list kosong

Saat p==ptr  
Berarti sudah sampai di akhir list

Dua syarat ini  
mungkinkah disatukan ?

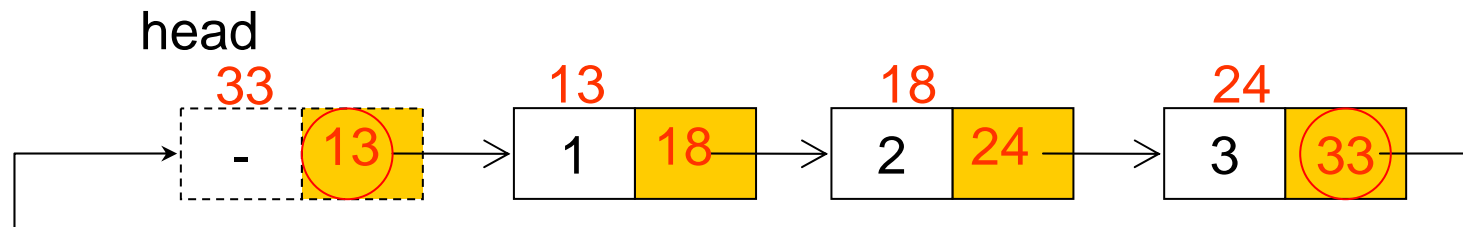
memakai head

# Circular List memakai head

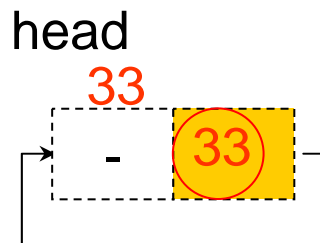


# Circular List memakai head

```
struct CELL head;
```



Circular List dengan 3 sel

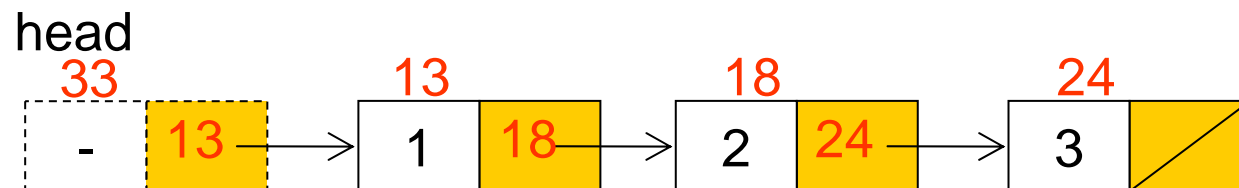


Circular List dalam kondisi kosong

`head.next == & head`

(pointer next dari head menunjuk ke head itu sendiri)

Catatan: bandingkan Circular List di atas dengan linked list berikut



# Circular List memakai head

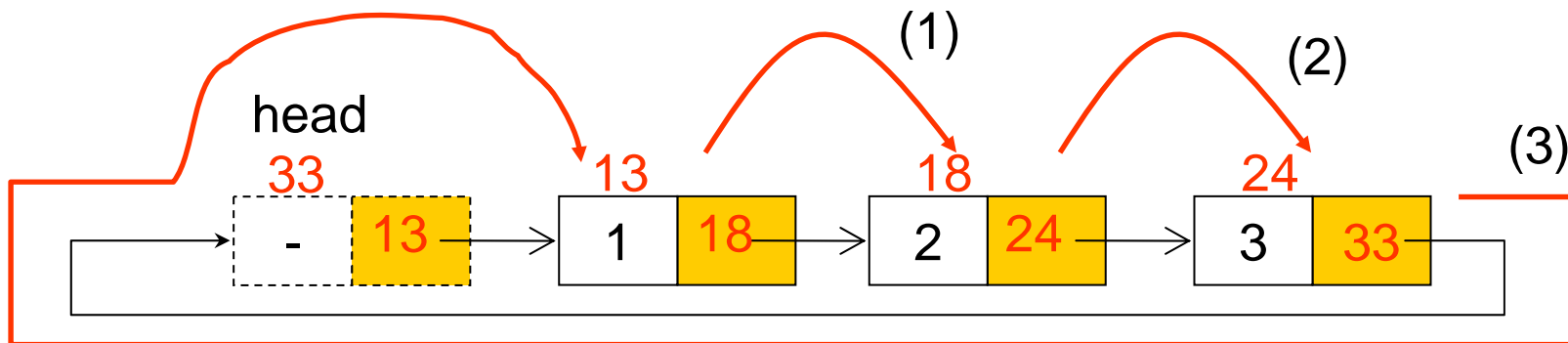
```
struct CELL head, *p;
```

```
for(p = head.next; p != &head; p = p->next) // head tidak di-lompati (skip)
```

```
{
```

perlu ditambahkan bagian untuk melewati "head" agar sel tsb.  
tidak diproses

```
}
```



# Beberapa Jenis Struktur Data

## 1. Array

1. Linear List
2. Stack
3. Queue

## 2. List

1. Linked List
2. Circular List
3. Bidirectional List
4. Multi list structure

## 3. Tree Structure

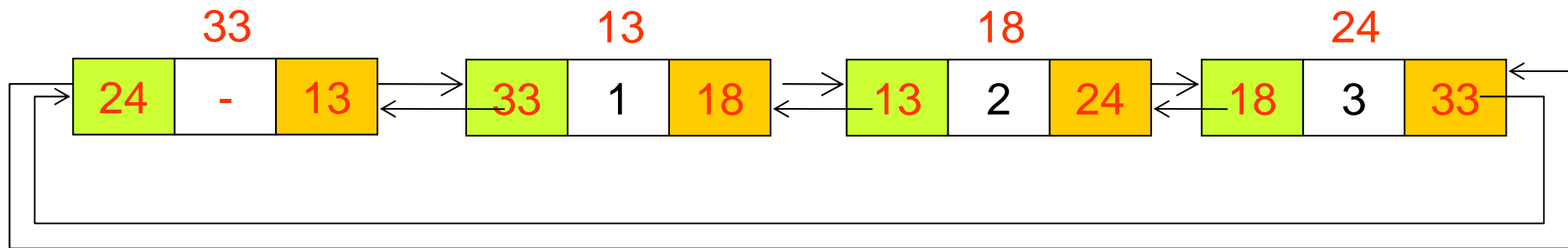
# Bidirectional List

# Apakah bidirectional list itu ?

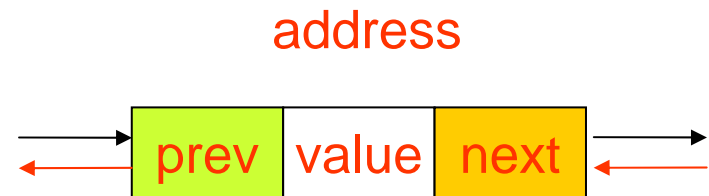
- Semua sel yang terdapat pada list disambungkan dengan pointer, sedangkan tiap sel memiliki TIGA komponen : value, pointer ke sel sebelumnya dan pointer ke sel berikutnya
- Dengan memiliki dua buah pointer ini, maka bidirectional list dapat diakses dengan DUA arah : ke arah depan dan ke belakang



# Bidirectional List



```
struct CELL {
    struct CELL *prev;
    struct CELL *next;
    MYDATA value;
};
```



Misalnya int, char, float, long, double, dsb

# Kelebihan dan kelemahan

## Kelebihan

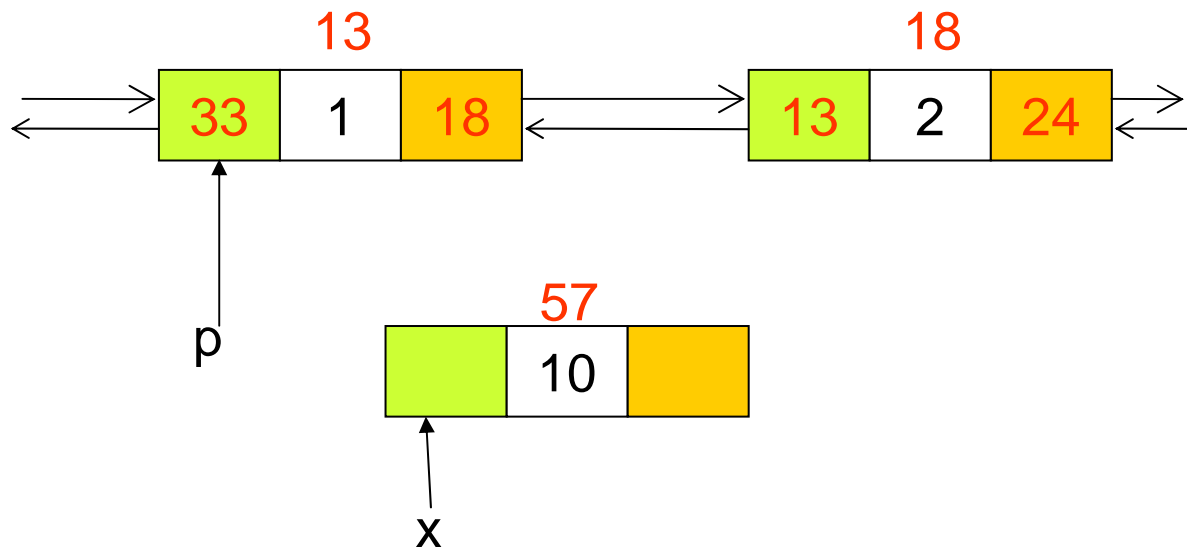
1. List bisa diakses dengan dua arah : ke depan maupun ke belakang
2. Penambahan dan penghapusan data menjadi mudah, karena pada tipe ini kita dapat menambahkan sel baru sesudah maupun sebelum sebuah sel. Demikian juga dalam hal menghapus sel.

## Kelemahan

1. Sebuah data memiliki dua buah pointer, sehingga memerlukan space yang lebih besar

# Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

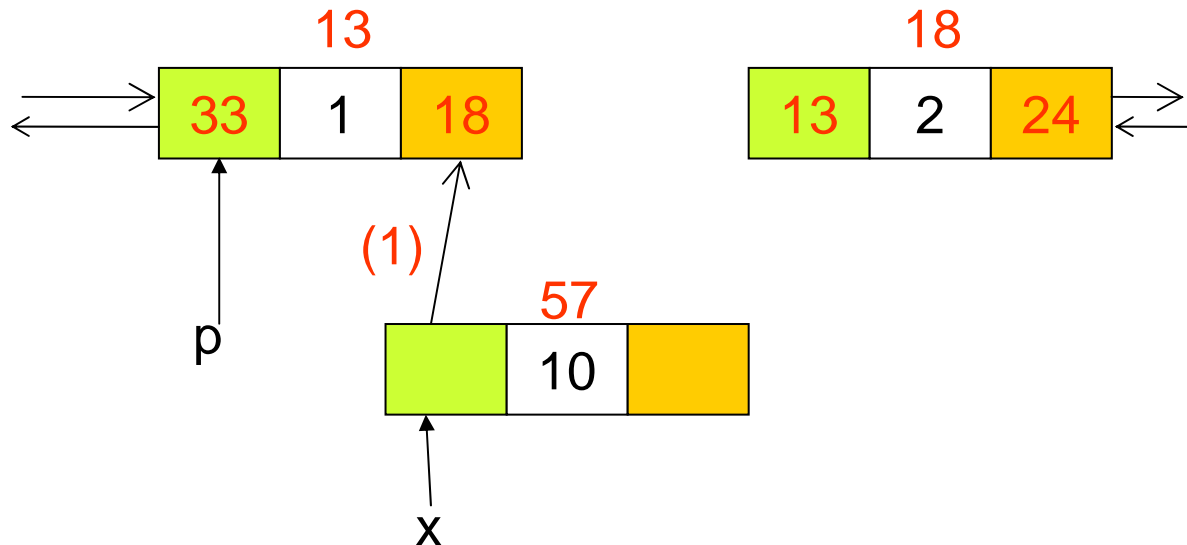


```
x->prev=p;  
x->next=p->next;  
p->next->prev=x;  
p->next=x;
```

- (1) Yang diubah bukan hanya
- (2) pointer next
- (3) tetapi juga pointer prev
- (4)

# Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

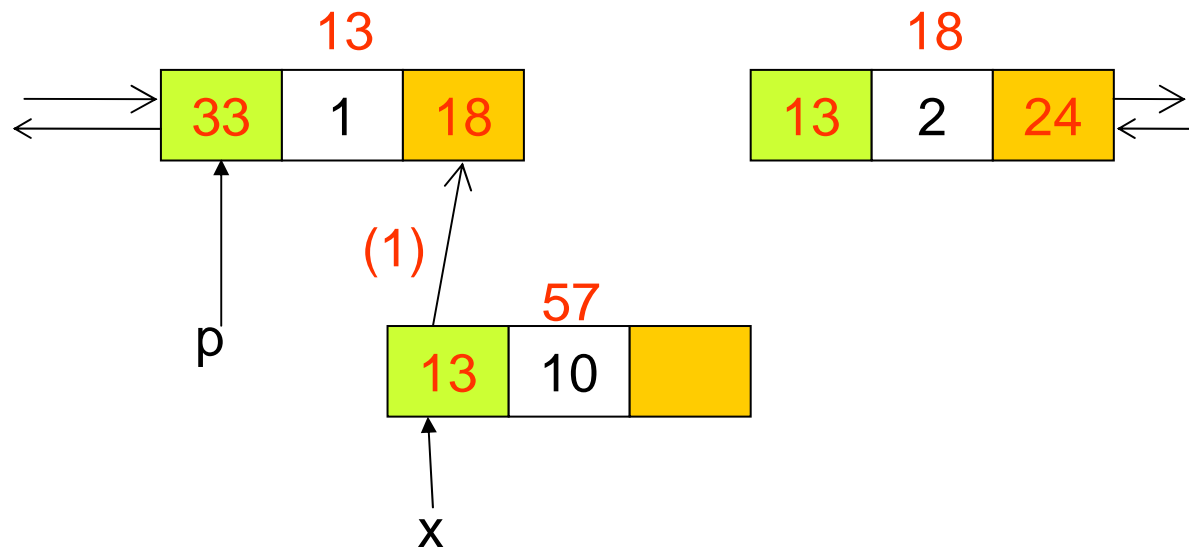


```
x->prev=p;  
x->next=p->next;  
p->next->prev=x;  
p->next=x;
```

- (1) Yang diubah bukan hanya
- (2) pointer next
- (3) tetapi juga pointer prev
- (4)

# Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

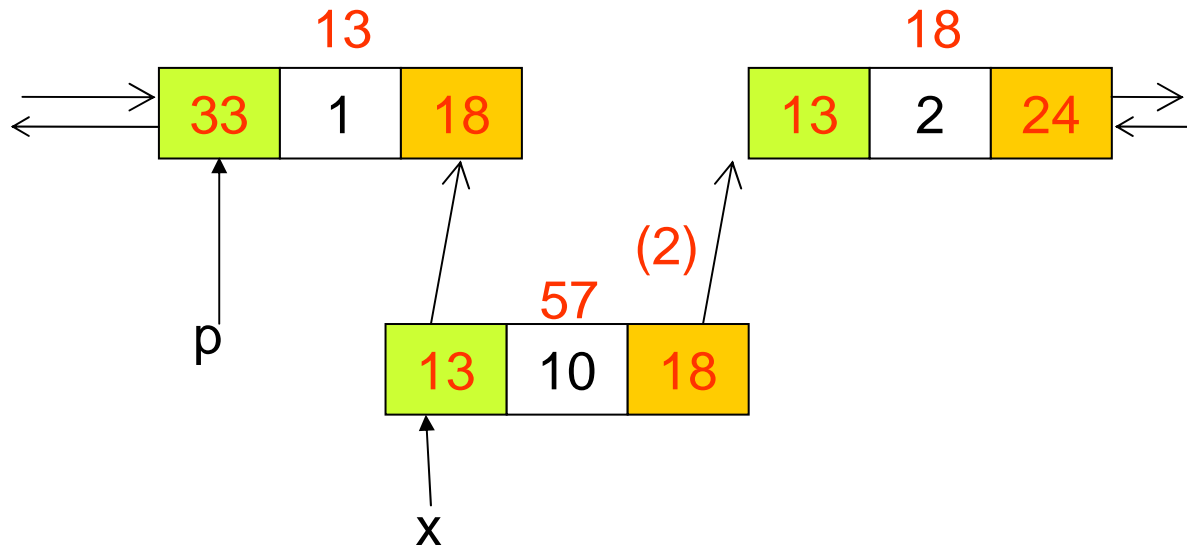


```
x->prev=p;  
x->next=p->next;  
p->next->prev=x;  
p->next=x;
```

- (1) Yang diubah bukan hanya
- (2) pointer next
- (3) tetapi juga pointer prev
- (4)

# Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

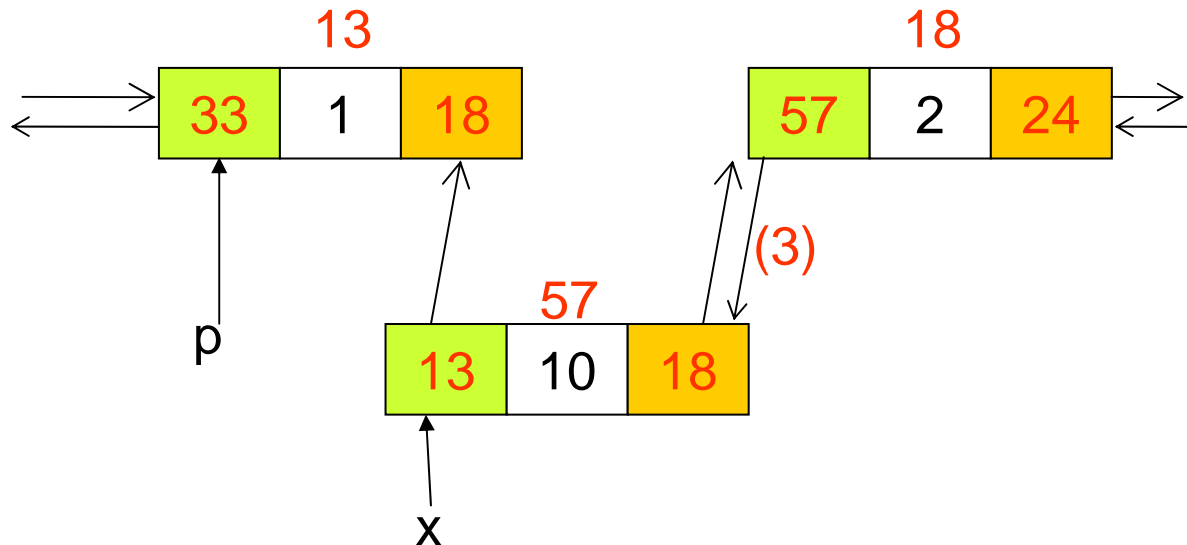


```
x->prev=p;  
x->next=p->next;  
p->next->prev=x;  
p->next=x;
```

- (1) Yang diubah bukan hanya
- (2) pointer next
- (3) tetapi juga pointer prev
- (4)

# Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p



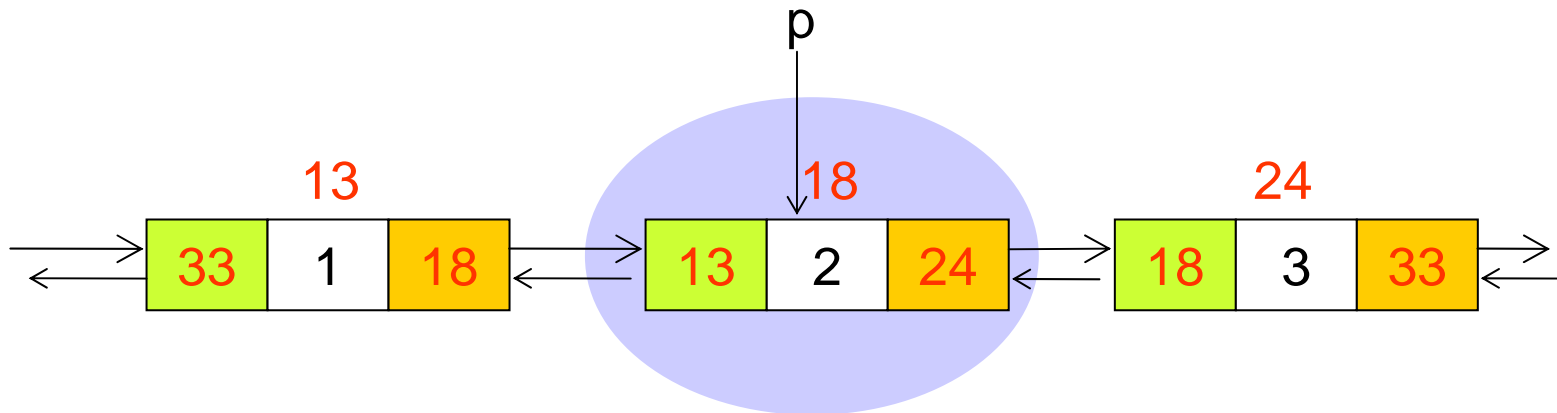
```
x->prev=p;  
x->next=p->next;  
p->next->prev=x;  
p->next=x;
```

- (1) Yang diubah bukan hanya
- (2) pointer next
- (3) tetapi juga pointer prev
- (4)



# Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p



```
p->prev->next=p->next; (1)
```

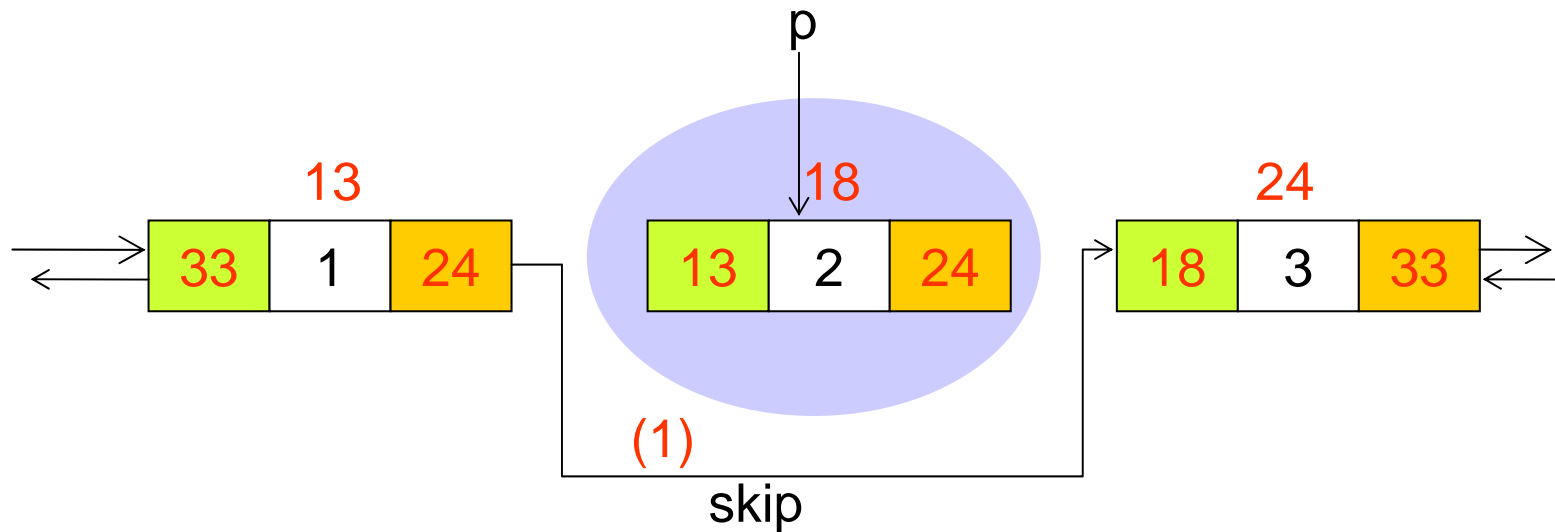
```
p->next->prev=p->prev; (2)
```

```
free(p);
```

Yang diubah bukan hanya pointer next  
tetapi juga pointer prev

# Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p



```
p->prev->next=p->next; (1)
```

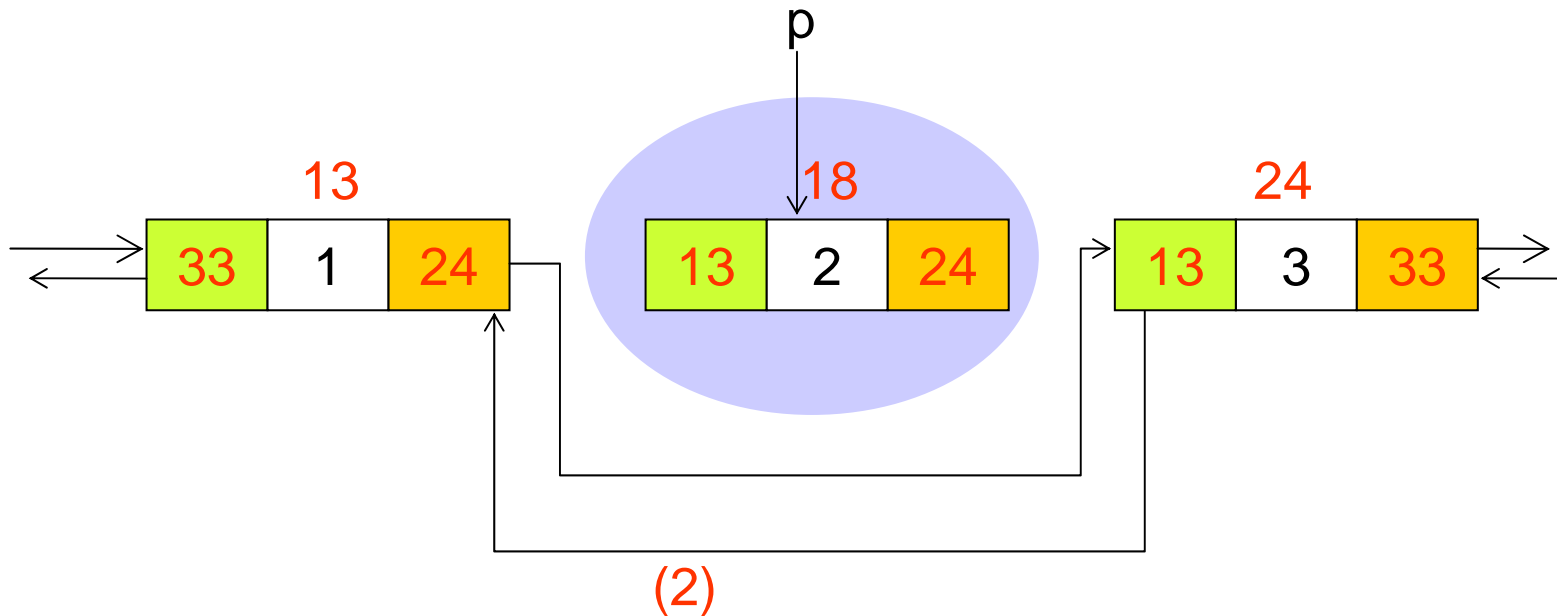
```
p->next->prev=p->prev; (2)
```

```
free(p);
```

Yang diubah bukan hanya pointer next  
tetapi juga pointer prev

# Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p



```
p->prev->next=p->next; (1)
```

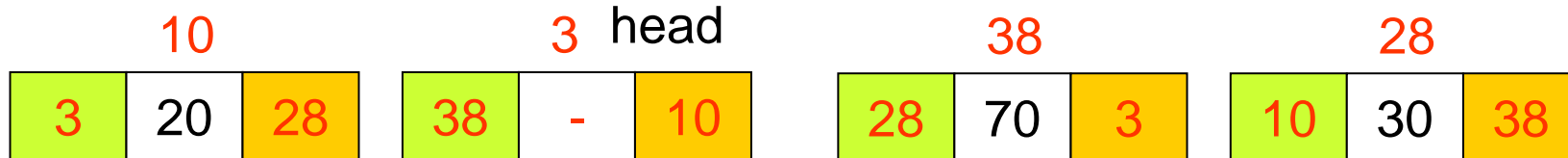
```
p->next->prev=p->prev; (2)
```

```
free(p);
```

Yang diubah bukan hanya pointer next  
tetapi juga pointer prev

# Latihan Bidirectional List

1. Gambarkan bidirectional list berikut



2. Tambahkan sel berikut setelah sel yang berisi value "20"



3. Gambarkan kondisi bidirectional list, setelah sel yang berisi value "70" dihapus. Kerjakan soal ini terhadap list yang diperoleh dari no.2 di atas.