04. Single Linked List

ARNA FARIZA

YULIANA SETIOWATI

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

Capaian Pembelajaran

1. Mahasiswa mengerti konsep alokasi memori secara dinamis menggunakan pointer.
2. Mahasiswa mengerti konsep single linked list dan operasi pada single linked list.
3. Mahasiswa dapat mengimplementasikan single linked list dalam bahasa pemrograman.

Materi

- Alokasi Memori
  - Fungsi malloc
  - Fungsi free
- Konsep Single Linked List
- Operasi pada Single Linked List
  - Operasi Menyisipkan data
  - Operasi Menghapus data
- Implementasi Stack dengan Single Linked List

Mengapa perlu Alokasi Memori

- Tipe data array mempunyai ukuran yang tetap
- Pada beberapa kasus, ukuran dari sebuah obyek tidak bisa dipastikan sampai dengan waktu dieksekusi *(run time)*
- Alokasi memori (memory allocation) memungkinkan untuk membuat ukuran buffer dan array secara dinamik → artinya ruang dalam memori akan dialokasikan ketika program dieksekusi.

Fungsi-Fungsi Alokasi Memory

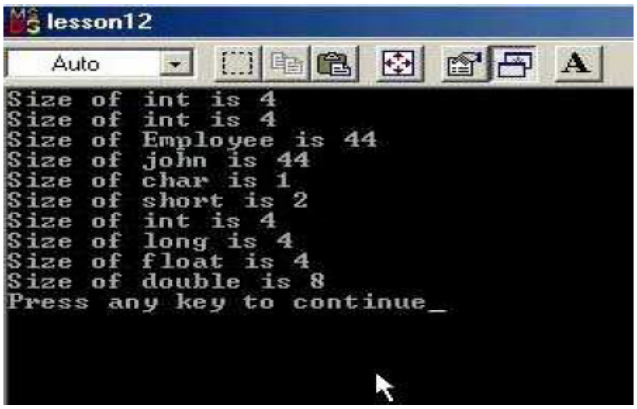
- *sizeof()*
- *malloc()*
- *free()*
- Berada pada library stdlib.h

sizeof()

- Untuk mendapatkan ukuran dari berbagai tipe data, variabel ataupun struktur.
- *Return value* : ukuran dari obyek yang bersangkutan dalam byte.
- Parameter dari *sizeof()* : sebuah obyek atau sebuah tipe data

```
typedef struct employee_st {
    char name[40];
    int id;
} Employee;
void main()
{
    int myInt;
    Employee john;
    printf("Size of int is %d\n",sizeof(myInt));
    printf("Size of int is %d\n",sizeof(int));
    printf("Size of Employee is %d\n",sizeof(Employee));
    printf("Size of john is %d\n",sizeof(john));
    printf("Size of char is %d\n",sizeof(char));
    printf("Size of short is %d\n",sizeof(short));
    printf("Size of int is %d\n",sizeof(int));
    printf("Size of long is %d\n",sizeof(long));
    printf("Size of float is %d\n",sizeof(float));
    printf("Size of double is %d\n",sizeof(double));
    return 0;
}
```


Output



malloc()

- Fungsi standar yang digunakan untuk mengalokasikan memori
- Format :

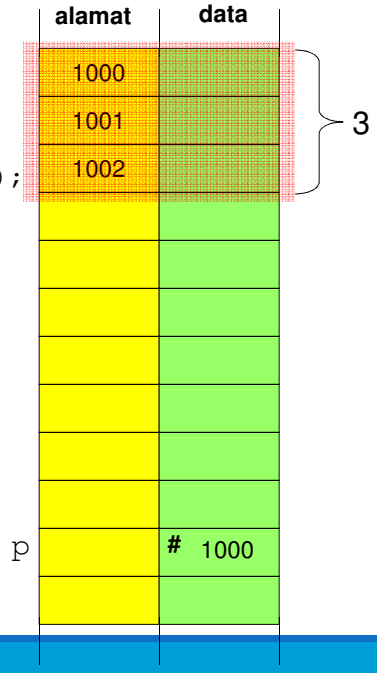
  *void *malloc(int jml_byte)*
- Banyaknya byte yang akan dipesan dinyatakan sebagai parameter fungsi.
- Return value dari fungsi ini adalah sebuah pointer yang tak bertipe *(pointer to void)* yang menunjuk ke buffer yang dialokasikan.
- Pointer tersebut haruslah dikonversi kepada tipe yang sesuai (dengan menggunakan *type cast*) agar bisa mengakses data yang disimpan dalam buffer.

malloc()

- Jika proses alokasi gagal dilakukan, fungsi ini akan memberikan return value berupa sebuah pointer NULL.
- Sebelum dilakukan proses lebih lanjut, perlu terlebih dahulu dipastikan keberhasilan proses pemesanan memori, sebagaimana contoh berikut:

```
int *x;
x = (int *) malloc(3 * sizeof(int));
if (x== NULL) {
    printf("Error on malloc\n");
    exit(0);
}
else {
    lakukan operasi memori dinamis…
}
```

```
int *p;
p=(int *) malloc(3 * sizeof(int));
```



Beberapa variasi malloc

```
char *nama;
nama=(char *) malloc (sizeof(char));
nama=(char *) malloc (20*sizeof(char));
char *nama[10];
nama[0]=(char *) malloc (20*sizeof(char));
…
nama[9]=(char *) malloc (20*sizeof(char));
```

```
typedef struct {
    int tanggal;
    int bulan;
    int tahun;
} TGL;
```

```
TGL *tgl_lahir;
tgl_lahir=(TGL *) malloc (sizeof(TGL));
tgl_lahir=(TGL *) malloc (20*sizeof(TGL);
TGL *tgl_lahirA[10];
tgl_lahirA[0]=(TGL *) malloc (2*sizeof(TGL));
…
tgl_lahirA[9]=(TGL *) malloc (2*sizeof(TGL));
```

free()

- Jika menggunakan memori yang dialokasikan secara dinamis, maka memori harus dibebaskan kembali untuk dikembalikan kepada sistem.
- Sehingga ruang tersebut bisa dipakai lagi untuk alokasi variabel dinamis lainnya.
- Format :

  *void free(void *pblok);*

- dimana pblok adalah pointer yang menunjuk ke memori yang akan dibebaskan.

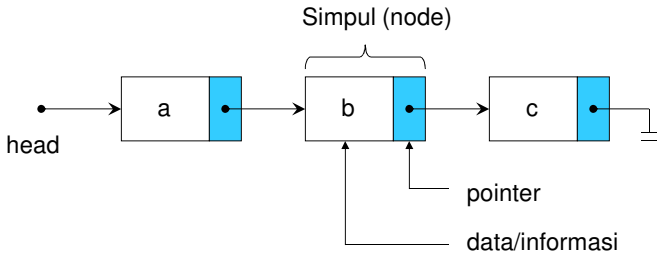
free()

```
#include <stdio.h>
#include <stdlib.h>
main()
{
   char *pblok;
   pblok = (char *) malloc(500 * sizeof(char));

   if (pblok == NULL)
    puts("Error on malloc");
   else {
    puts("OK, alokasi memori sudah dilakukan");
    puts("------");
    free(pblok);
    puts("Blok memori telah dibebaskan kembali");
   }
 }
```


Linked List

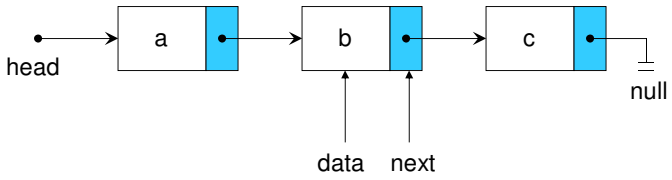
- Senarai berantai (linked list) : penyimpanan data dari sekelompok data yang diorganisasi secara dinamis dengan urutan tertentu



Single Linked List

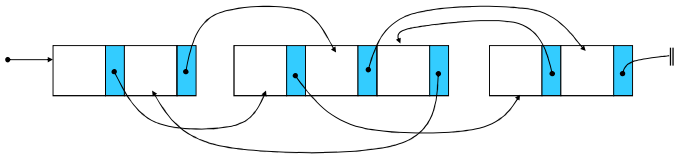
- Terdiri dari elemen-elemen individu yang dihubungkan dengan pointer tunggal
- Masing-masing elemen terdiri dari dua bagian, yaitu bagian data/informasi yang disimpan dan bagian pointer yang disebut dengan pointer *next*
- Pointer *next* pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list
- Elemen awal diakses oleh pointer head

Contoh Single Linked List



Alokasi Memori pada Single Linked List

- Elemen-elemen linked list dialokasikan secara dinamis (menggunakan *malloc*)
- Lokasinya terpencar-pencar di memori
- Pointer menjamin semua elemen dapat diakses



Deklarasi Simpul pada Single Linked List

```
typedef struct simpul Node;
struct simpul {
   int data;
   Node *next;
};
```

Variabel *head* dan *baru*

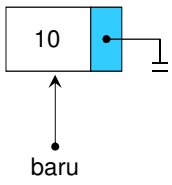
- *head* adalah variabel pointer yang menunjuk ke awal list
- *baru* adalah variabel pointer yang menunjuk ke simpul baru

```
Node *head = NULL;
Node *baru;
```


Alokasi Simpul Baru

```
baru=(Node *) malloc (sizeof(Node));
baru->data = x;
baru->next = NULL;
```

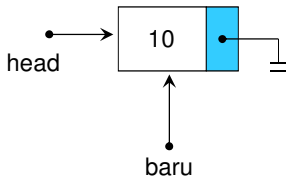
Jika x=10, maka



Membentuk Simpul Awal

- *head* menunjuk awal list, karena hanya ada satu simpul maka head menunjuk baru.

```
head = baru;
```

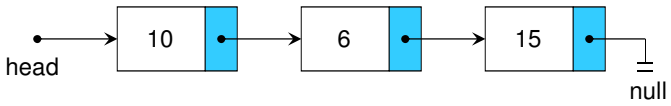


Operasi pada Single Linked List

1. Mencetak Simpul
2. Menyisipkan Simpul
3. Menghapus Simpul

Operasi Mencetak Simpul

- Operasi mencetak simpul dimulai dari posisi head, dan mencetak data setiap simpul



Mencetak Single Linked List

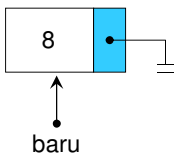
```
Node *p = head;
while (p!= NULL){
   printf("%d ", p->data);
   p = p->next;
}
printf("\n");
```

Operasi Menyisipkan Simpul

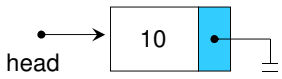
- Operasi menyisipkan simpul terdiri dari:
  - Sisip awal list
  - Sisip akhir list
  - Sisip sebelum simpul tertentu
  - Sisip setelah simpul tertentu

Sisip Awal List

Buat simpul baru:

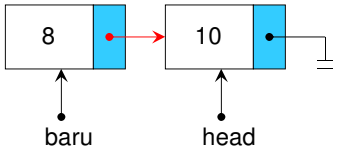


Linked list:



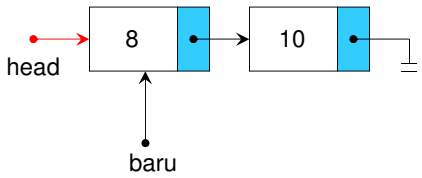
Sisip Awal List

1. *baru->next* menunjuk simpul *head*



Sisip Awal List

2. *head* menunjuk *baru*

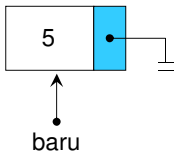


Sisip Awal List

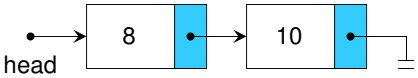
```
baru->next = head;
head = baru;
```


Sisip Akhir List

Buat simpul baru:

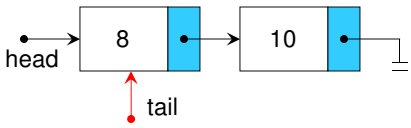


Linked list:



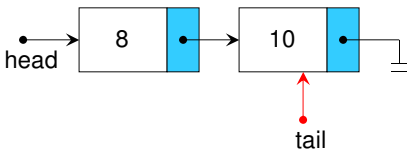
Sisip Akhir List

1. *tail* menunjuk *head*



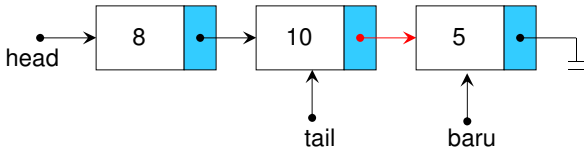
Sisip Akhir List

2. Arahkan *tail* pada akhir list



Sisip Akhir List

3. *tail->next* menunjuk *baru*

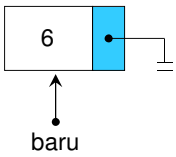


Sisip Akhir List

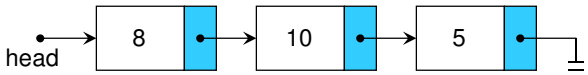
```
Node *tail = head;
while(tail->next != NULL)
   tail = tail->next;
tail->next = baru;
```

Sisip Sebelum Simpul x (misal: x=5)

Buat simpul baru:

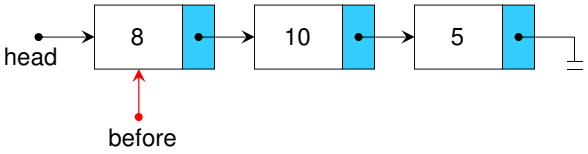


Linked list:



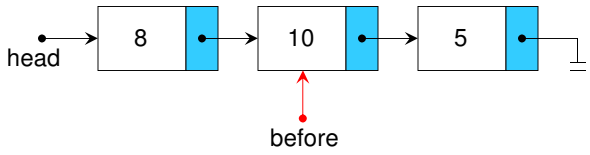
Sisip Sebelum Simpul x (misal: x=5)

1. *before* menunjuk *head*



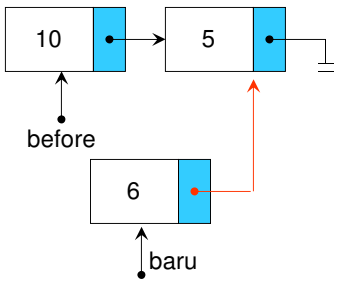
Sisip Sebelum Simpul x (misal: x=5)

2. Arahkan *before* pada simpul sebelum x = 5



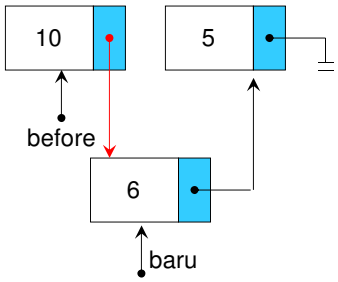
Sisip Sebelum Simpul x (misal: x=5)

3. *baru->next* menunjuk *before->next*



Sisip Sebelum Simpul x (misal: x=5)

4. *before->next* menunjuk *baru*

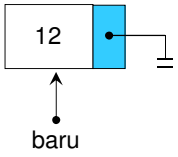


Sisip Sebelum Simpul Tertentu

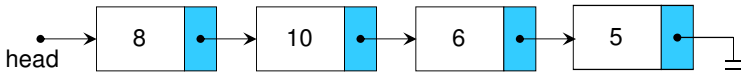
```
Node *before = head;
while(before->next->data != x)
   before = before->next;
baru->next = before->next;
before->next = baru;
```

Sisip Setelah Simpul x (misal: x=10)

Buat simpul baru:

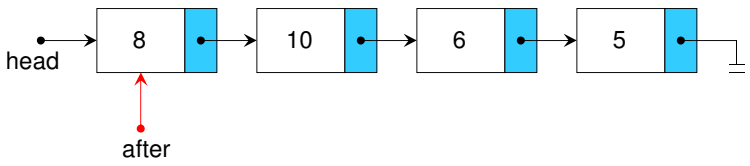


Linked list:



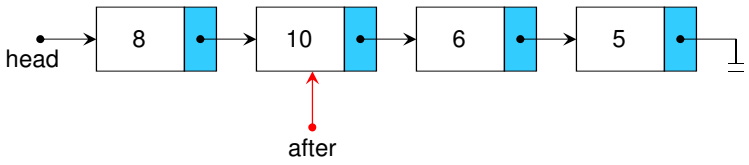
Sisip Setelah Simpul x (misal: x=10)

1. *after* menunjuk *head*



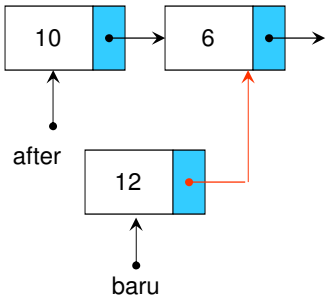
Sisip Setelah Simpul x (misal: x=10)

2. Arahkan *after* pada simpul x = 10



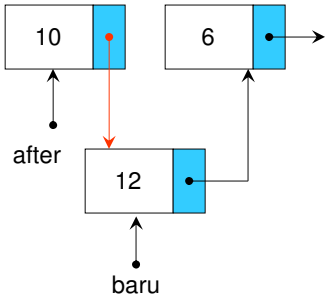
Sisip Setelah Simpul x (misal: x=10)

3. *baru->next* menunjuk *after->next*



Sisip Setelah Simpul x (misal: x=10)

4. *after->next* menunjuk *baru*



Sisip Sebelum Simpul Tertentu

```
Node *after = head;
while(after->data != x)
   after = after->next;
baru->next = after->next;
after->next = baru;
```

Operasi Menghapus Simpul

- Operasi menghapus simpul terdiri dari:
  - Hapus simpul awal
  - Hapus simpul akhir
  - Hapus simpul tertentu

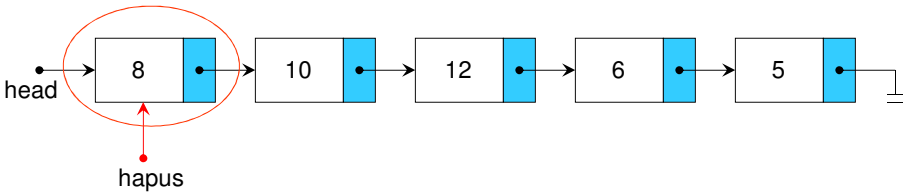
Fungsi free_Node

- Sebelum menghapus simpul, buat fungsi untuk membebaskan alokasi memori dengan fungsi *free*

```
void free_Node(Node *p)
{
   free(p);
   p=NULL;
}
```

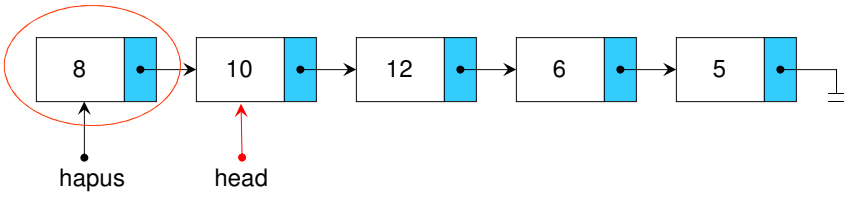
Hapus Simpul Awal

1. *hapus* menunjuk simpul yang sama dengan *head*



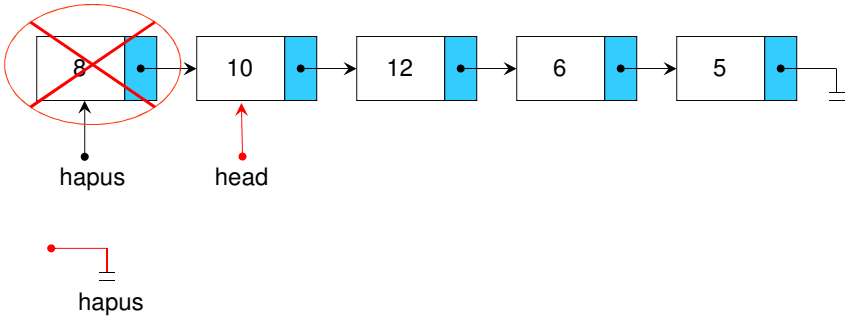
Hapus Simpul Awal

2. *head* menunjuk *hapus->next*



Hapus Simpul Awal

3. *free_node(hapus)*

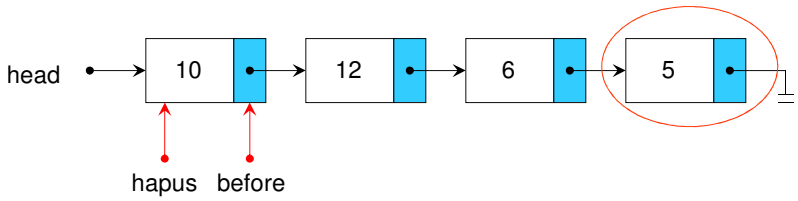


Hapus Simpul Awal

```
Node *hapus = head;
head = hapus->next;
free_Node(hapus);
```

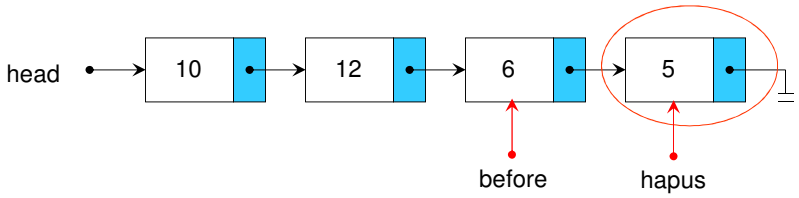
Hapus Simpul Akhir

1. *hapus* dan *before* menunjuk *head*



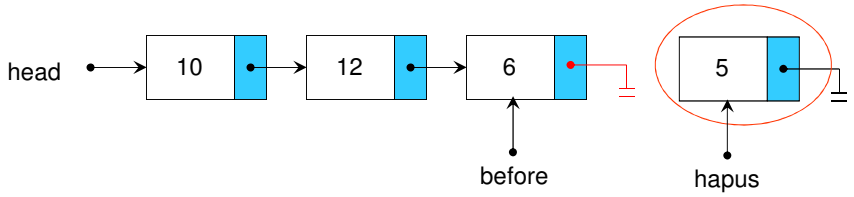
Hapus Simpul Akhir

2. Arahkan *hapus* ke akhir list dan *before* ke node sebelumnya



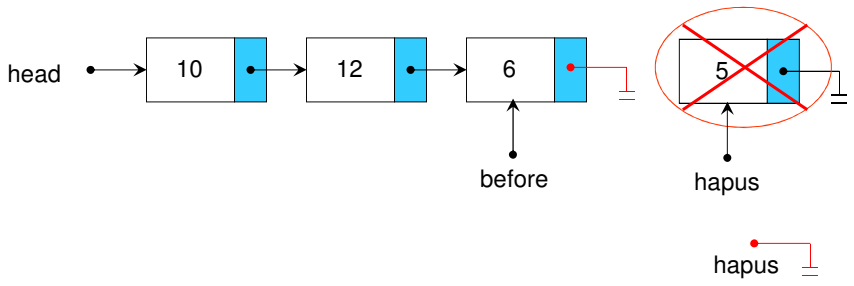
Hapus Simpul Akhir

3. *before->next* menunjuk NULL



Hapus Simpul Akhir

4. *free_Node(hapus)*

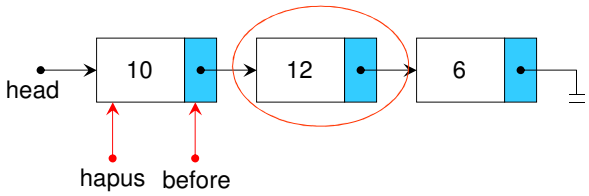


Hapus Simpul Akhir

```
Node *hapus = head, before;
while(hapus->next != NULL){
   before = hapus;
   hapus = hapus->next;
}
before->next = NULL;
free_Node(hapus);
```

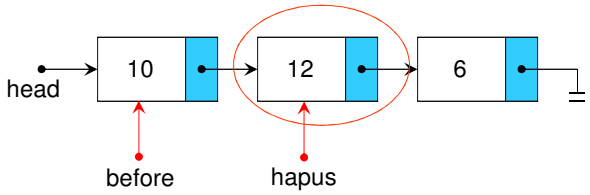

Hapus Simpul Tengah (misal x = 12)

1. *hapus* dan *before* menunjuk *head*



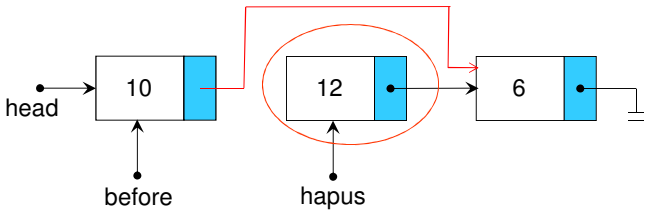
Hapus Simpul Tengah (misal x = 12)

2. Arahkan *hapus* ke simpul 12 dan *before* ke simpul sebelumnya



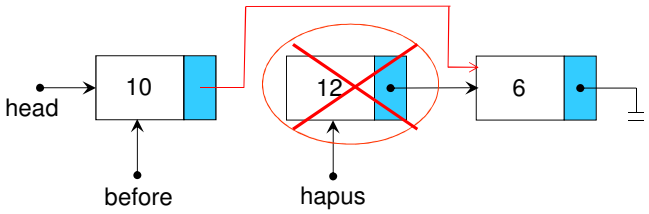
Hapus Simpul Tengah (misal x = 12)

3. *before->next* menunjuk ke *hapus->next*



Hapus Simpul Tengah (misal x = 12)

4. *free_Node(hapus)*



Hapus Simpul Tengah

```
Node *hapus = head, before;
while(hapus->data != x){
   before = hapus;
   hapus = hapus->next;
}
before->next = hapus->next;
free_Node(hapus);
```

Implementasi Stack dengan Single Linked List

- Stack adalah penyimpanan data dengan konsep Last In First Out (LIFO)
- Terdapat satu penunjuk yaitu *front*
- Fungsi Push menambah simpul pada posisi *front*
- Fungsi Pop menghapus simpul pada posisi *front*

Deklarasi Stack dengan Single Linked List

```
typedef struct simpul Node;
typedef int itemtype;
struct simpul {
   itemtype item;
   Node *next;
};
typedef struct {
   Node *front;
   int count;
} Stack;
```

Fungsi pada Stack

- Inisialisasi → *front* menunjuk NULL, *count*=0
- Kosong → jika *front* = NULL
- Enqueue → buat simpul, sisip awal list (posisi *front*)
- Dequeue → hapus awal list (posisi *front*)

Fungsi Inisialisasi

```
void inisialisasi (Stack *s)
{
   s->front = NULL;
   s->count = 0;
}
```

Fungsi Kosong

```
int Kosong (Stack *s)
{
   return(s->front==NULL)
}
```

Fungsi Push

```
void Push (Stack *s, itemtype x)
{
    Node *baru = (Node *) malloc (sizeof(Node));
    if(baru==NULL) {
        printf("Alokasi gagal\n");
        exit(1);
    }
    else {
        baru->item = x;
        baru->next = s->front;
        s->front = baru;
        s->count++;
    }
}
```

Fungsi Pop

```
itemtype Pop (Stack *s)
{
    Node *hapus;
    itemtype temp;
    if(Kosong(s)) {
        printf("Stack kosong\n");
        return ` `;
    }
    else {
        temp = s->front->item;
        hapus = s->front;
        s->front = hapus->next;
        free(hapus);
        s->count--
        return temp;
    }
}
```

Rangkuman

- Simpul pada single linked list terdiri dari bagian data dan pointer *next*
- Operasi pada single linked list terdiri dari operasi cetak, sisip dan hapus
- Operasi cetak dapat dilakukan dari *head* sampai akhir list
- Operasi sisip terdiri dari sisip awal list, sisip akhir list, sisip setelah simpul tertentu, sisip sebelum simpul tertentu
- Operasi hapus terdiri dari hapus awal list, hapus akhir list dan hapus simpul tertentu
- Implementasi Stack dengan single linked list, pada operasi Push dengan sisip awal list dan pada operasi Pop dengan hapus awal list

Latihan

1. Buatlah single linked list dengan data bertipe integer yang dapat melakukan operasi sisip secara terurut dan hapus simpul tertentu dengan ketentuan sebagai berikut :
   a) Operasi sisip, buat sebuah simpul baru,
      - Jika data simpul baru < data pada head, maka gunakan sisip awal list
      - Jika pencarian data simpul baru mencapai NULL (data belum ada), sisip akhir list
      - Jika data simpul baru = data pada simpul tertentu maka berikan pesan simpul sudah ada (duplikat)
      - Lainnya sisip sebelum simpul tertentu
   b) Operasi hapus,
      - Jika posisi simpul yang dihapus pada head, gunakan hapus awal list
      - Jika posisi simpul yang dihapus pada tail, gunakan hapus akhir list
      - Lainnya hapus simpul tengah

Latihan

2. Implementasikan Stack dengan single linked list, buatlah menu Push, Pop dan Tampil
3. Buatlah single linked list dengan simpul berupa data mahasiswa yang terdiri dari NRP, Nama dan Kelas. Buatlah operasi Sisip secara terurut, Hapus data mahasiswa tertentu dan Update data (nama dan kelas saja)