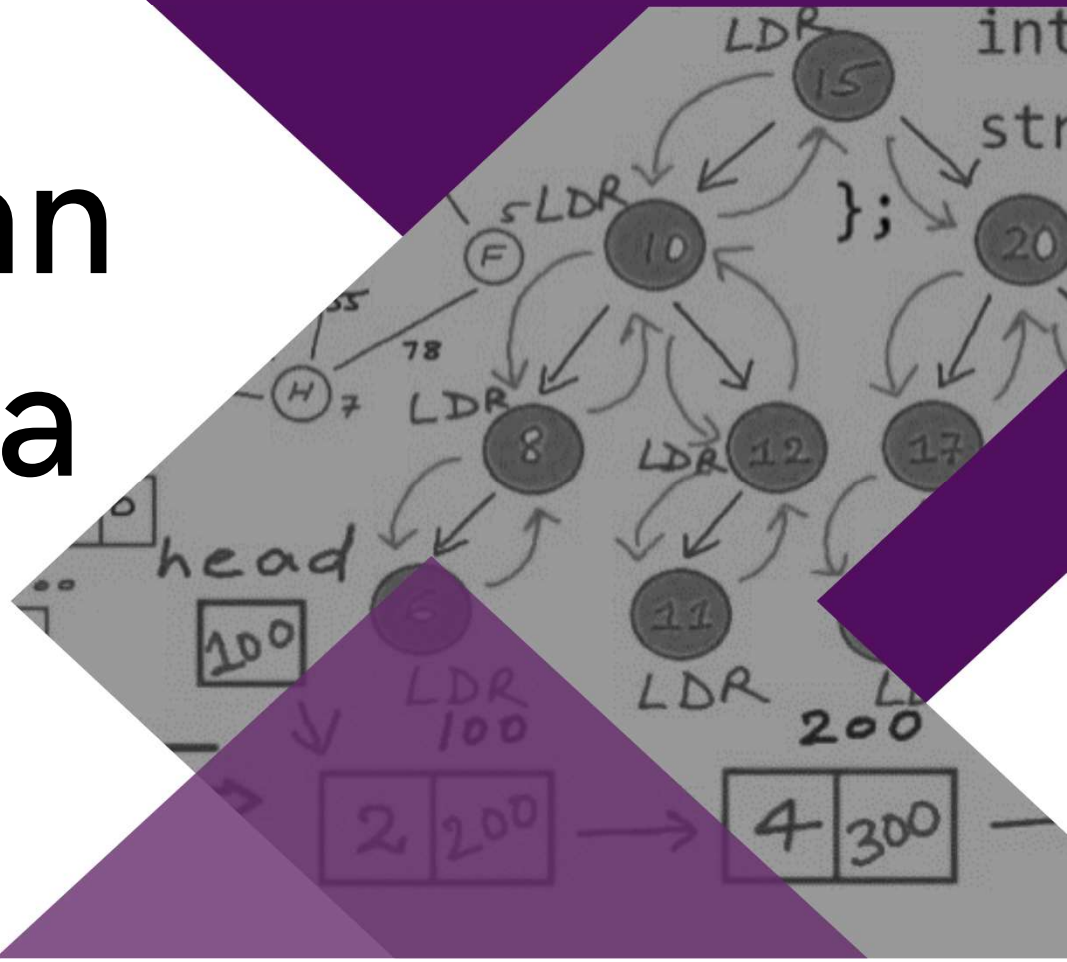
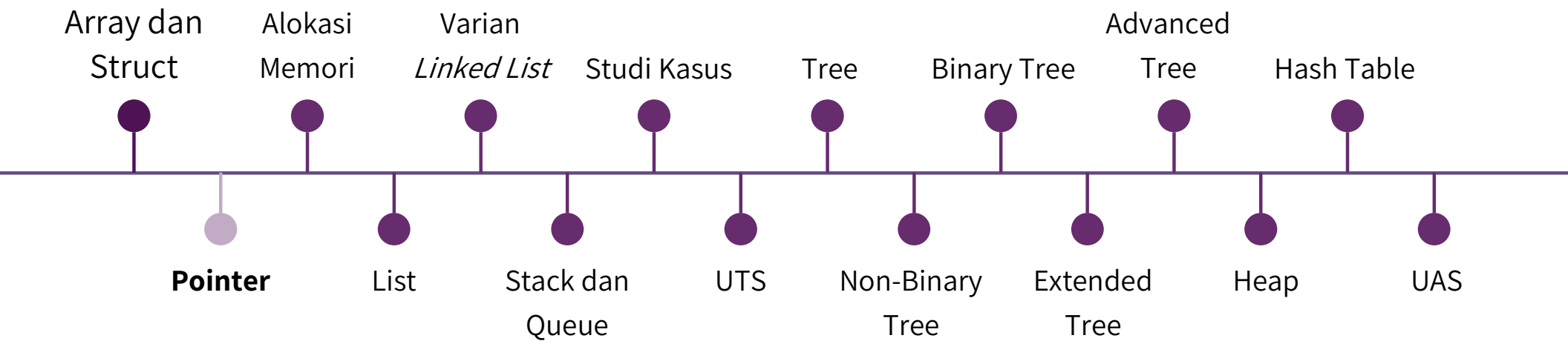
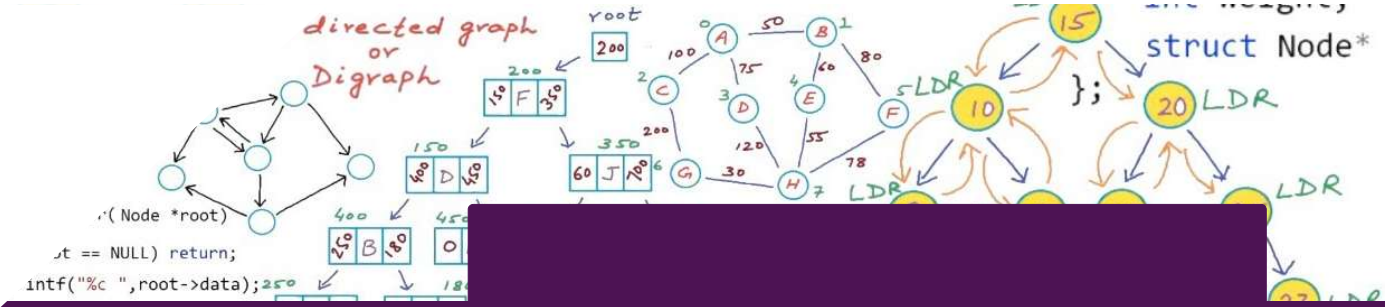
# Algoritma dan Struktur Data



# Pekan 2


Array dan Struct
Pointer
Alokasi Memori
List
Varian *Linked List*
Stack dan Queue
Studi Kasus
UTS
Tree
Non-Binary Tree
Binary Tree
Extended Tree
Advanced Tree
Heap
Hash Table
UAS

# Tujuan




| | |
|---|---|
| 1 | Mahasiswa memahami konsep pointer dan *indirect addressing* |
| 2 | Mahasiswa mampu menggunakan variable pointer dengan tepat. |
| 3 | Mahasiswa mengetahui struktur data *linked-list*. |
| 4 | Mahasiswa mampu menggunakan pointer untuk menghubungkan struktur data. |

# Pointer

# Pointer (Variable Penunjuk)

A memory cell that stores the address of a data item.

# Deklarasi Pointer

```
tipe_data *nama_variabel;
```

```
int m = 25;
int *itemp;

itemp = &m;
```

# Inisiasi Pointer

```
nama_var_pointer = &nama_var_diacu;
```

```
int m = 25;
int *itemp;

itemp = &m;
```

# *Indirect Reference*

Accessing the contents of a memory cell through a pointer variable that stores its address.

```
int m = 25;
int *itemp;

itemp = &m;

*itemp = 35;
cout << *itemp;

*itemp = 2 * (*itemp)
```

# Variabel Biasa dan Pointer

**Variable Biasa**

- Berisi nilai/data
- Operasi aritmatika
- Direct access

**Variable Pointer**

- Berisi alamat memori suatu variable
- Indirect access (operator & untuk menunjuk alamat, operator * untuk mengakses nilai dari variabel yang ditunjuk)

# Pointer dan Reference

Bahasa pemrograman yang berbeda menyediakan mekanisme yang berbeda untuk melakukan tugas pointer, dan tidak semuanya menyediakan alamat memori mentah kepada pemrogram.

Bahasa pemrograman tingkat rendah seperti C dan C++ memberikan raw pointer dan memungkinkan untuk secara langsung mengakses lokasi memori yang mereka simpan.

Bahasa pemrograman lainnya, seperti Python, menggunakan referensi, yang menggunakan sintaks seperti variabel biasa tetapi masih memungkinkan untuk merujuk variabel lain.

Variasi-variasi yang berbeda ini datang dengan perilaku dan penggunaan yang berbeda.

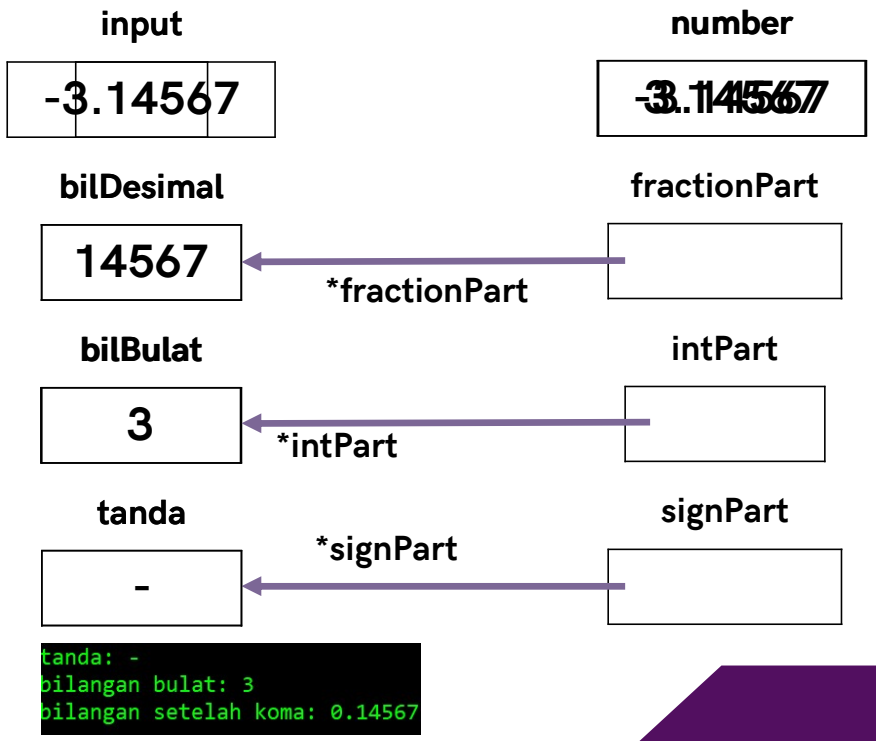
Istilah pointer dalam mata kuliah ini  mencakup semua variabel yang diimplementasikan oleh pointer maupun referensi.

# Kegunaan Pointer

# Pointer ke File

# Pointer sebagai parameter output

```cpp
#include <iostream>

using namespace std;
void pisah(float number, char *signPart, int *intPart, float *fractionPart);

int main()
{
    float input, bilDesimal;
    int bilBulat;
    char tanda;

    cin >> input;
    pisah(input, &tanda, &bilBulat, &bilDesimal);
    cout << "tanda: " << tanda << endl;
    cout << "bilangan bulat: " << bilBulat << endl;
    cout << "bilangan setelah koma: " << bilDesimal << endl;

    return 0;
}

void pisah(float number, char *signPart, int *intPart, float *fractionPart)
{
    if (number > 0)
        *signPart = '+';
    else if (number == 0)
        *signPart = ' ';
    else
    {
        *signPart = '-';
        number = number * -1;
    }

    *intPart = number;
    *fractionPart = number - *intPart;
}
```



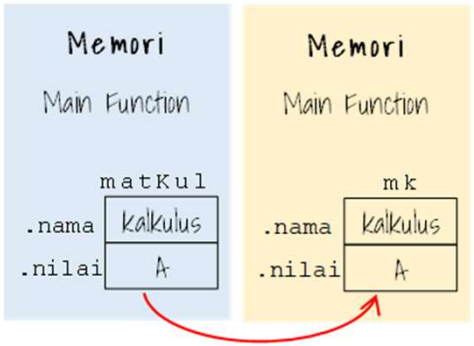
# Pointer sebagai representasi array dan string

- Saat mendeklarasikan array sebagai sebuah parameter formal untuk subprogram, pada saat subprogram dipanggil nilai parameter actual tidak disalin oleh subprogram. Pemanggil mengirimkan alamat variable yang digunakan pada parameter actual.
- Pada bagian realisasi subprogram walaupun nama array berbeda, jika kita mengakses indeks elemen tertentu kita akan mengakses alamat variabel yang sama dengan pemanggilnya.

# Pointer sebagai representasi array dan string

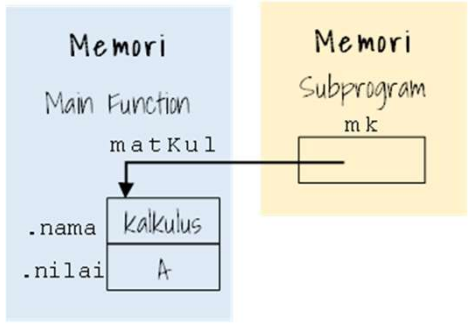
```
#include <iostream>

using namespace std;
void tambah(int larik1[3],int larik2[3], int larik3[]);

int main()
{
    int m1[3] = {2,1,4};
    int m2[3] = {9,15,26};
    int hasil[3];

    tambah (m1,m2,hasil);
    for (int i=0;i<3;i++)
        cout << hasil[i] << endl;

    return 0;
}

void tambah(int larik1[3], int larik2[3], int larik3[3])
{
    for (int i=0;i<3;i++)
        larik3[i] = larik1[i] + larik2[i] ;
}
```

# Pointer untuk Parameter Input Berupa Tipe Data Bentukan

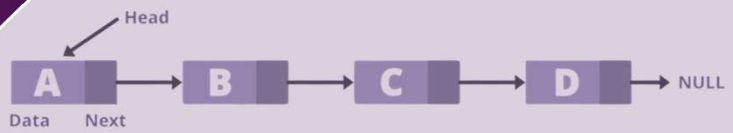



```cpp
#include <iostream>
using namespace std;

 typedef struct {
    string nama;
    char nilai;
}mataKuliah;

void cetak_matakuliah(mataKuliah mk);

int main()
{
    mataKuliah matkul;

    matkul.nama = "Kalkulus";
    matkul.nilai = 'A';

    cetak_matakuliah(matkul);

    return 0;
}

void cetak_matakuliah(mataKuliah mk)
{
    cout << mk.nama << endl;
    cout << mk.nilai<< endl;
}
```

```cpp
#include <iostream>
using namespace std;

 typedef struct {
    string nama;
    char nilai;
}mataKuliah;

void cetak_matakuliah(mataKuliah *mk);

int main()
{
    mataKuliah matkul;

    matkul.nama = "Kalkulus";
    matkul.nilai = 'A';

    cetak_matakuliah(&matkul);

    return 0;
}

void cetak_matakuliah(mataKuliah *mk)
{
    cout << mk->nama << endl;
    cout << mk->nilai << endl;
}
```



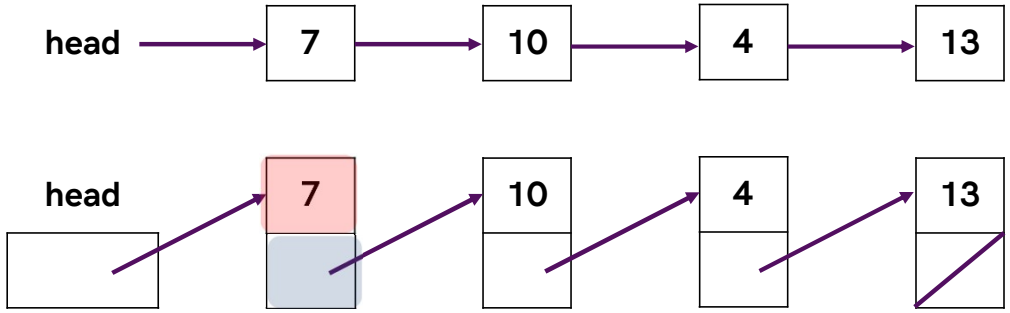
# *Linked List*



# *Linked List*

A linear data structure in which elements, called nodes, are connected together using pointers.

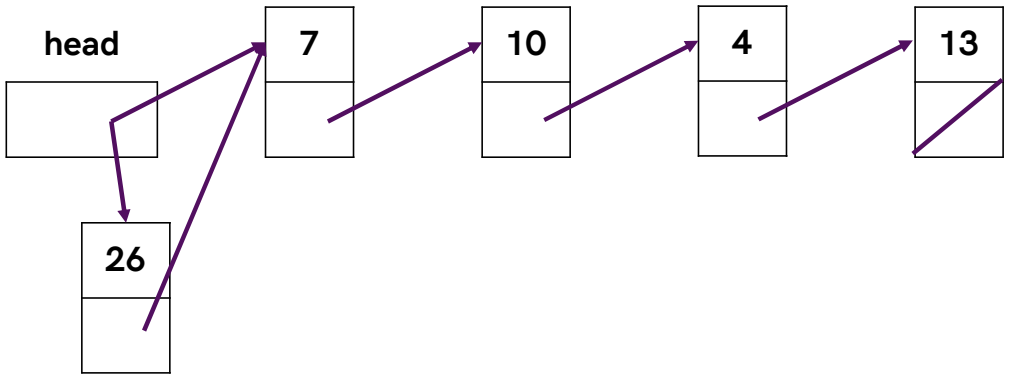
Each node consists of two parts: **data** and a **pointer** to the next node in the sequence.



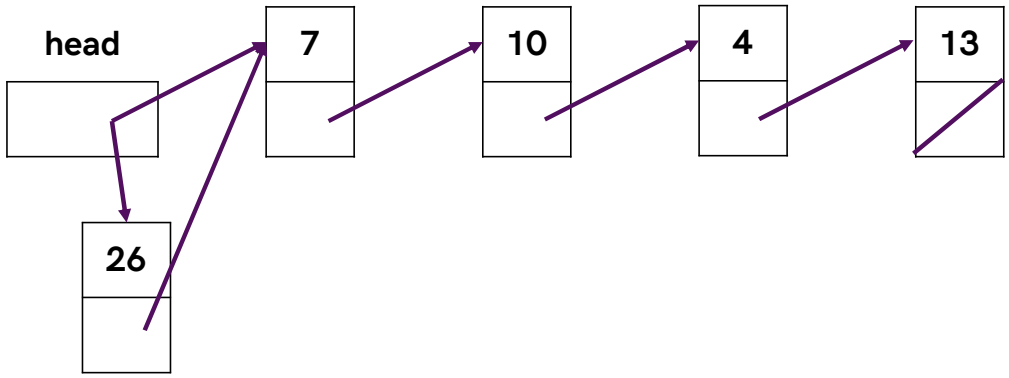
# Operasi pada *Linked List*

| | |
|---|---|
| `PushFront(key)` | Add to front |
| `TopFront()` | Return front item |
| `PopFront()` | Remove front item |
| `PushBack(Key)` | Add to back |
| `TopBack()` | Return back item |
| `PopBack()` | Remove back item |
| `Find(Key)` | Is key in list? |
| `Erase(Key)` | Remove key from list |
| `Empty()` | Empty list? |
| `AddBefore(Node,Key)` | Adds key before node |
| `AddAfter(Node,Key)` | Adds key after node |

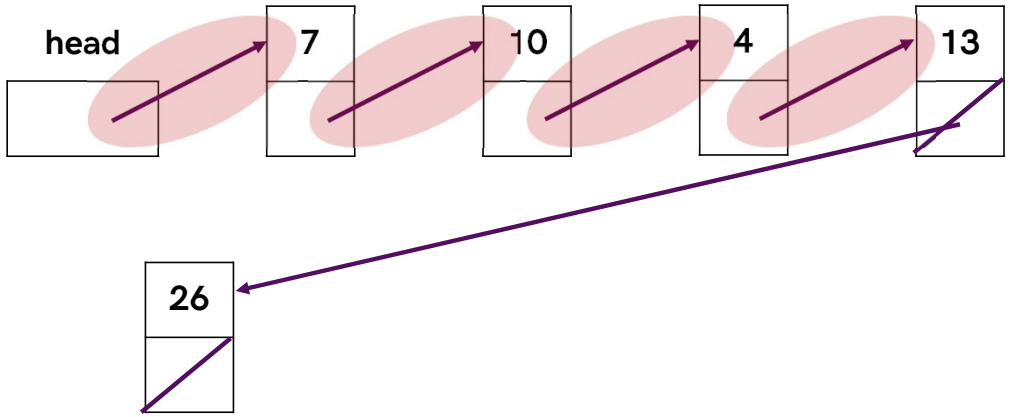
# PushFront



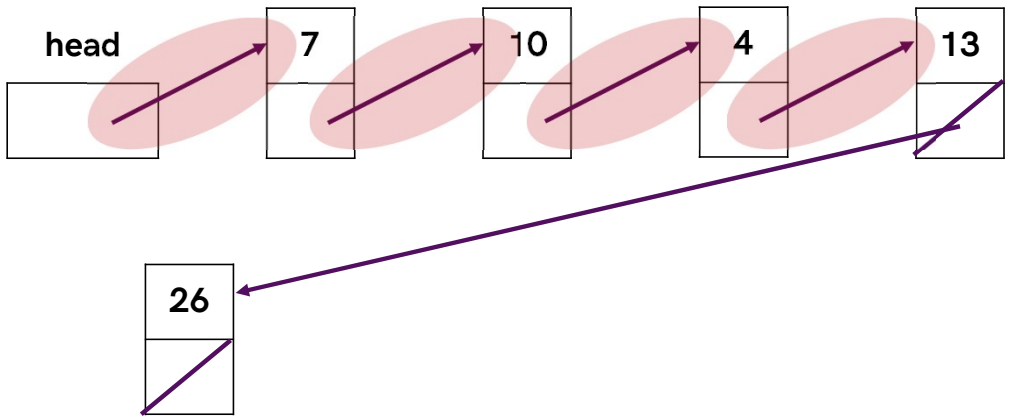
# PopFront



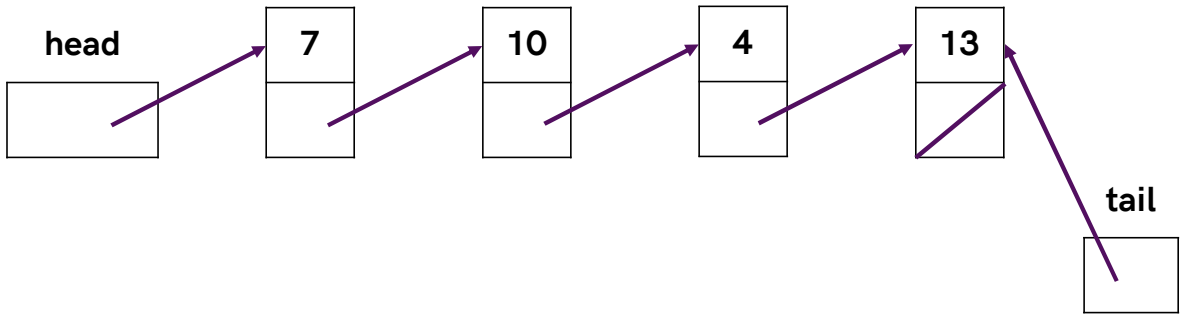
# PushBack



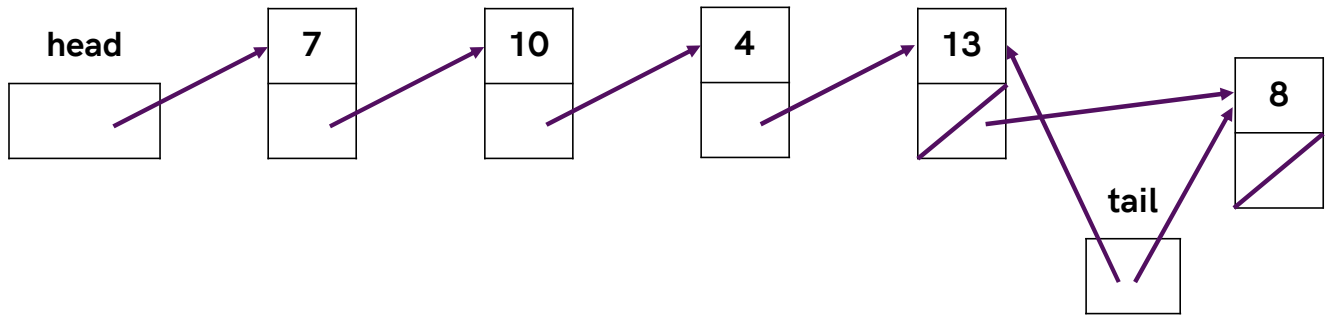
# PopBack



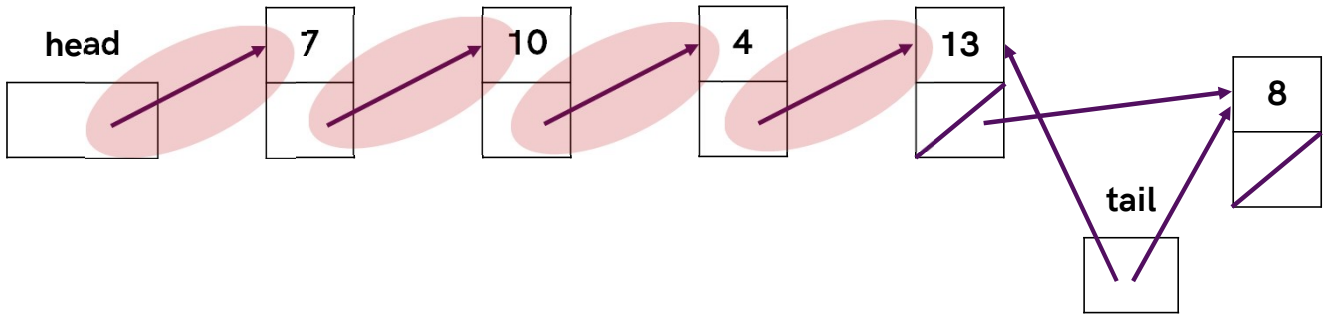
# *Linked List* dengan Pointer *Tail*



# PushBack dengan Pointer *Tail*

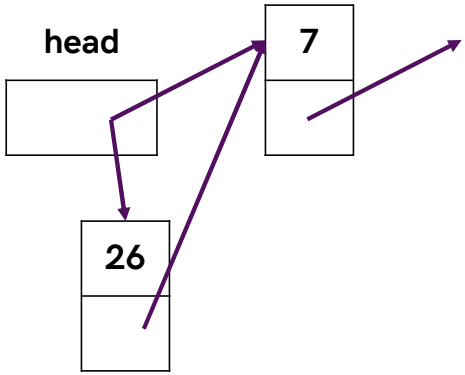


# PopBack dengan Pointer *Tail*



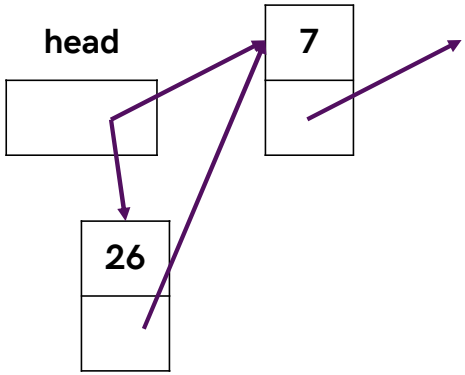
# PushFront

```
PushFront(key)
node ←new node
node.key ← key
node.next ← head
head ← node
if tail = nil:
    tail ← head
```



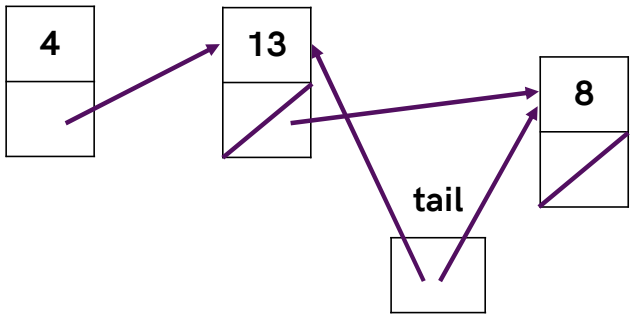
# PopFront

```
PopFront()

if head = nil:
   ERROR: empty list
head ← head.next
if head = nil:
   tail ← nil
```



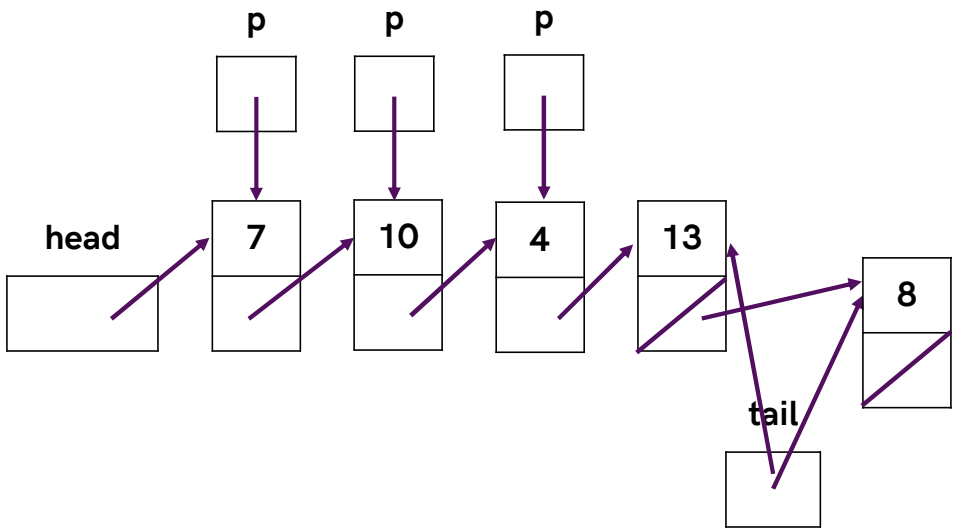
# PushBack

```
PushBack(key)

node ←new node
node.key ← key
node.next =nil
if tail = nil:
    head ← tail ← node
else:
    tail.next ← node
    tail ← node
```



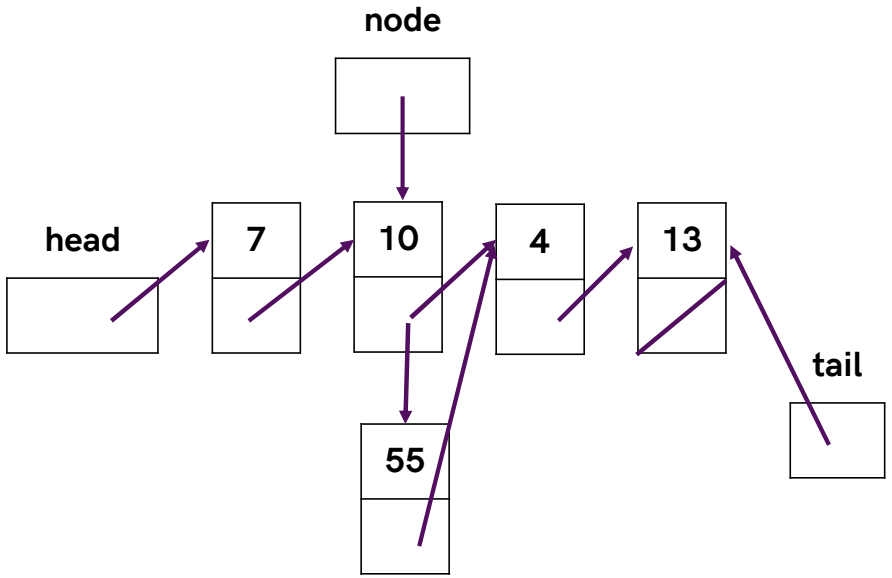
# PopBack

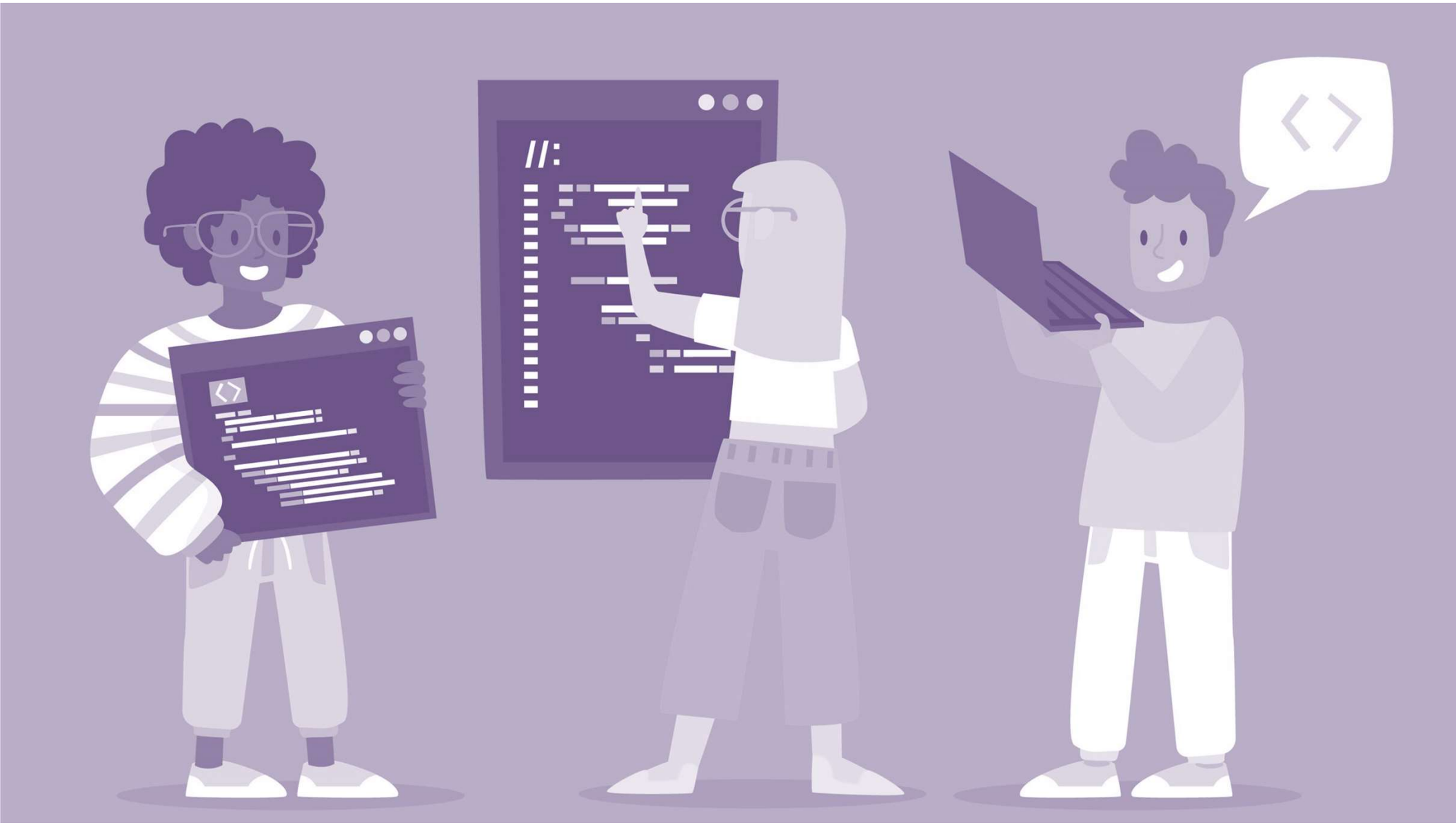
```
PopBack()

if head = nil:   ERROR: empty list
if head = tail:
    head ← tail ←nil
else:
    p ← head
    while p.next.next ≠ nil:
        p ← p.next
    p.next ← nil; tail ← p
```



# AddAfter

```
AddAfter(node, key)
node2 ←new node
node2.key ← key
node2.next = node.next
node.next = node2
if tail = node:
    tail ← node2
```





```cpp
#include <iostream>
using namespace std;

int main()
{
    int m = 10, n = 5;
    int* mp, *np;
    mp = &m;
    np = &n;
    *mp = *mp + *np;
    *np = *mp - *np;

    cout << m << " " << *mp << " " << n << " " << *np;

    return 0;
}
```