# *Circular Linked-List*

(TIB11 – Struktur Data)

Pertemuan 13, 14

# Sub-CPMK

- Mahasiswa mampu membuat *Circular Linked-List* dan mengakses data nya (C3, A3)

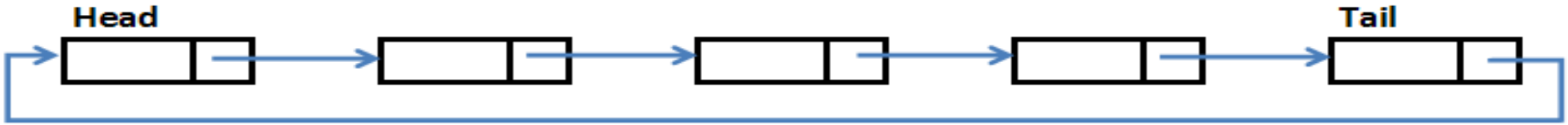
## Materi:

1. Konsep *Circular Linked-List*
2. Menambahkan Node
3. Mencari Node
4. Menghapus Node
5. Memindahkan Node

# 1. Konsep *Circular Linked-List*

# 1.1. *Circular Linked List*

- *Next pointer* pada *tail*, menunjuk ke *Head*



# 1.2. *Circular Linked List* yang lain

- *Double Linked List* Dapat menjadi *circular linked List* dengan cara mengarahkan *next link* pada *tail* ke *head* dan mengarahkan *previous link* pada *head* ke *Tail*

# 1.3. Operasi pada *Circular Linked List*

- Sama seperti pada *Linked List* sebelumnya, *Circular linked list* juga memiliki operasi-operasi cari, sisip/tambah dan hapus dengan memperhatikan operasi jika dilakukan pada *Head* atau *Tail* agar *linked list* tetap circular

# 2. Menambahkan Node

# 2.1. *Insert Operation*

- Pada bagian depan *list*

  Sama seperti pada *single Linked-list*

- Pada bagian tengah *list*

  Sama seperti pada *single Linked-list*

- Pada bagian akhir *list*

  Perbedaan operasi sisip antara *single* LL dengan *Circular* LL adalah pada penentuan *next* dari node baru sebagai node terakhirnya. Jika pada *single* LL *next link* node baru di isi dengan NULL, maka pada *Circular* LL *link* node baru diisi dengan *Head*

# 2.2. *Insert* Pada Bagian Akhir *List*

Perbedaan operasi sisip antara *single* LL dengan *Circular* LL adalah pada penentuan *next* dari node baru sebagai node terakhirnya. Jika pada *single* LL *next link* node terakhir di isi dengan NULL, maka pada *Circular* LL *link* node terakhir diisi dengan *Head*

- Buat sebuah node baru

  Baru = malloc()

- Isi informasi node baru tersebut
- *Set next link* dari node Baru ke *Head*

  Baru->next = *Head*

- Arahkan *next link* pada *last* node atau *tail* ke node baru

  *Tail->next* = Baru

# 2.3. Keistimewaan *Insert Operation* pada *Circular* LL

- Sebenarnya mengingat sifat *circular linked-list* yang *link* nya akhirnya kembali ke *link* awal, penambahan Node pada *circular linked list* dapat dilakukan dengan dengan cara yang sama seperti menambah node di tengah *list* pada *single Linked-List* tanpa membedakan penambahan pada *Head*, tengah atau *Tail*

# 3. Mencari Node

- Untuk mencari Node pada *Circular* LL memiliki keistimewaan, kita dapat menyimpan Alamat *current* node (node yang ditunjuk oleh *pointer* Ptr) ke suatu variabel *pointer*, kemudian lakukan pelacakan node satu persatu diawali dari *current* node sampai ditemui *next link* dari node yang diakses menunjuk ke current node atau sampai data yang dicari ditemukan.

- Contoh:

```
Head = Ptr; //perhatikan!
While (Ptr->next != Head)
OR (Ptr->dat != cari) do
{
   Ptr = Ptr->next
}
```

# 3.1. Mencari Node Dengan Cara Seperti *Single* LL

- Pencarian node Masih dapat dilakukan dengan cara yang sama dengan *single* LL juga dengan Pembatasan pencarian sampai ditemui Ptr->*next* = *Head*
- Contoh:

```
Ptr = Head;
While (Ptr->next != Head) OR (Ptr->dat != cari) do
{
  Ptr = Ptr->next
}
```

# 4. Menghapus Node

# 4.1. *Delete Operation*

- Pada bagian depan / *delete head*

  Proses sama seperti pada *single* LL
- Pada bagian tengah

  Proses sama seperti pada *single* LL
- Pada bagian akhir / *delete tail*

  Perbedaan operasi hapus antara *single* LL dengan *Circular* LL adalah pada penentuan *next* dari node sebelum node terakhir sebagai pengganti node terakhir yang akan dihapus.

  Jika pada *single* LL *next link* node sebelum node terakhir di isi dengan NULL, maka pada *Circular* LL link node sebelum node terakhir diisi dengan *Head*

# 4.2. *Delete* Pada Akhir *List*

- Lokasikan *pointer* ptrHapus ke node yang akan dihapus dan Ptr ke node sebelum node terakhir yang akan dihapus
- Isi *next link Pointer* Ptr dengan *Head* (dapat juga dilakukan dengan node berikut dari node terakhir yang tentu saja menunjuk ke *Head* juga)

  Ptr->*next* = *Head* atau Ptr->*next* = ptrHapus->*next*
- Hapus *current* node

  free(ptrHapus)

# 4.3. Keistimewaan *Delete Operation* pada *Circular* LL

- Sebenarnya mengingat sifat *circular linked-list* yang *link* nya akhirnya kembali ke *link* awal, penghapusan Node pada *circular linked list* dapat dilakukan dengan dengan cara yang sama seperti menghapus node di tengah *list* pada single *Linked-List* tanpa membedakan penghapusan pada *Head*, tengah atau *Tail*

# 5. Memindahkan Node

Mengingat sifat *circular linked-list* yang *link* nya akhirnya kembali ke *link* awal, pemindahkan Node pada circular *linked list* dapat dilakukan dengan dengan cara yang sama seperti memindahkan node dari tengah ke posisi tengah yang lain pada single *Linked-List* tanpa membedakan penghapusan pada *Head*, tengah atau *Tail*

# 5.1. Varian Linked List Lainnya

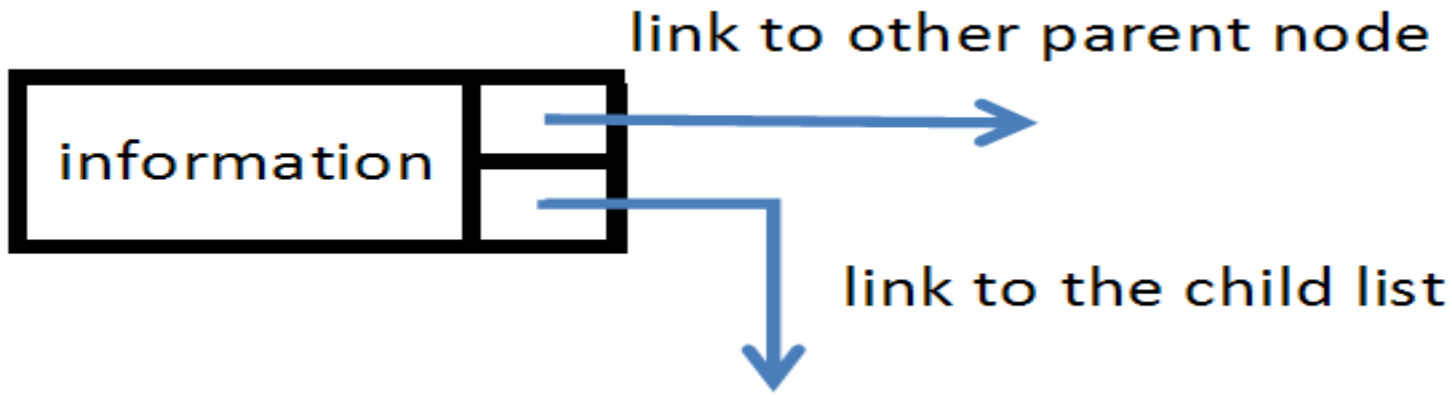
- *Linked List* selain *single Linked-List*, *double Linked List* maupun *circular linked-list* dapat dibentuk menjadi *linked list* lainnya seperti *Multiple Linked List*, *Multilevel Linked List*, maupun *Tree* Dengan operasi-operasi yang khusus diterapkan pada bentuk-bentuk *list* tersebut

# 5.1.1. *Multilevel List*

- *List* sebagai *group list* dimana ada node yang menjadi *parent* dari suatu *groups* memiliki *extra link* untuk menunjuk ke list lain sebagai *child list* di samping *link* ke *next group list* node.
- *Element*
  - *Information*
  - *Link* ke node *parent* lain
  - *Link* ke *child list*

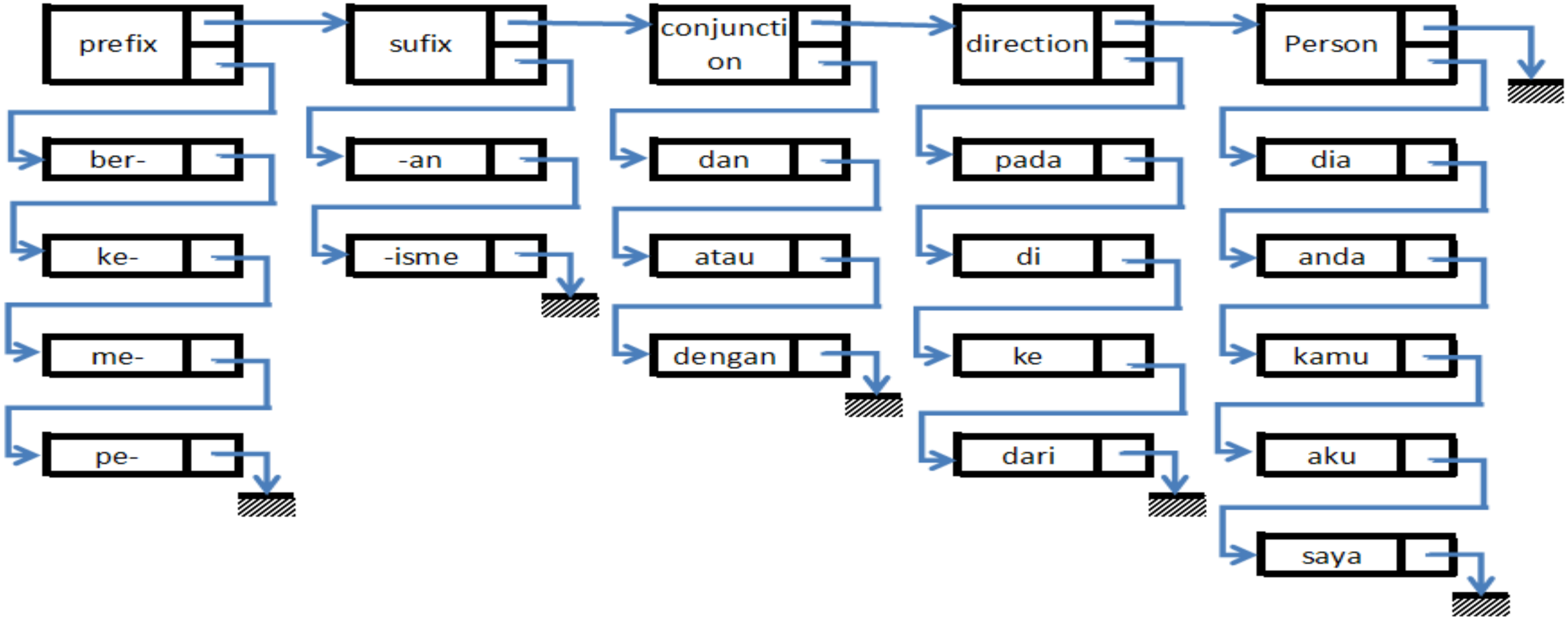
# 5.1.1. *Multilevel List* (Lanj.)

*Multilevel List Element*



# 5.1.1. *Multilevel List* (Lanj.)

## Contoh multilevel list



# 5.2. Catatan

- Contoh pada slide sebelumnya merupakan contoh dari penggunaan *multilevel list* pada aplikasi pemrosesan kata.
- *List* pertama merupakan *list* syntax, *link* kedua merupakan *list vocabulary* pada masing-masing syntax

# Ringkasan

- Penggunaan *Circular Linked List* cukup praktis, semua dapat dilakukan seperti operasi menambah, menghapus dan memindahkan list dari tengah ke posisi tengah yang lain dari single LL
- Untuk mencari Node pada *Circular* LL memiliki keistimewaan, kita dapat menyimpan Alamat *current* node (node yang ditunjuk oleh *pointer* Ptr) ke suatu variabel *pointer*, kemudian lakukan pelacakan node satu persatu diawali dari *current* node sampai ditemui *next link* dari node yang diakses menunjuk ke *current* node atau sampai data yang dicari ditemukan.

# Contoh Program Circular LL

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <string.h>
//Contoh Circular LL
//Record Definition
struct TheCell
{
  int dat;
  struct TheCell *berikut; //NEXT
};
```

# Contoh Program Circular LL (lanjutan-1)

```
//program utama
void main()
{
  int i;
  struct TheCell *tempPtr=NULL;  //sebagai penampung sementara
  struct TheCell *kepala=NULL;  //sebagai HEAD
  struct TheCell *ptrCell=NULL; //sebagai POINTER / PENUNJUK
```

# Contoh Program Circular LL (lanjutan-2)

```
for (i=1;i<=10;i++)
{
  tempPtr = (struct TheCell *) malloc(sizeof(struct TheCell));
  if (kepala == NULL)
  {
    //Mengisi Linked List dari keadaan kosong
    kepala = tempPtr;
    ptrCell = kepala;
    ptrCell->berikut = kepala;
  }
```

# Contoh Program Circular LL (lanjutan-3)

```
    else
    {
      tempPtr = ptrCell;
      //Mengisi pada Linked List yang sudah terbentuk
      tempPtr->berikut=ptrCell->berikut
      ptrCell->berikut = tempPtr;
      ptrCell = tempPtr;
    }
    //Isi Data
    ptrCell->dat=i*10;
}
```

# Contoh Program Circular LL (lanjutan-4)

```
  //Menampilkan Isi Linked List dari awal
  //Memastikan Linked List tidak kosong, yg ditandai dengan kepala != NULL
  if (kepala!=NULL)
  {
    cout<<"cell: {Alamat} [ Data | Cell berikut]"<<endl;
    //Arahkan ptrCell ke alamat yang ditunjuk oleh kepala
    ptrCell = kepala;
    do
    {
      //Tampilkan isi Node / Simpul
      cout<<"cell: {"<<ptrCell<<"} ["<<ptrCell->dat<<" | "<<ptrCell->berikut<<"]"<<endl;
      //Arahkan ptrCell ke Node/Simpul berikutnya
      ptrCell = ptrCell->berikut;
    }while (ptrCell != kepala); //keluar dari loop ketika ptrCell kembali ke kepala
    cout<<endl<<endl;
  }
  getch();
}
```

*Terimakasih*

# *TUHAN Memberkati Anda*

Teady Matius Surya Mulyana (tmulyana@bundamulia.ac.id)