FAKULTAS
TEKNOLOGI
INFORMASI
Universitas Advent Indonesia

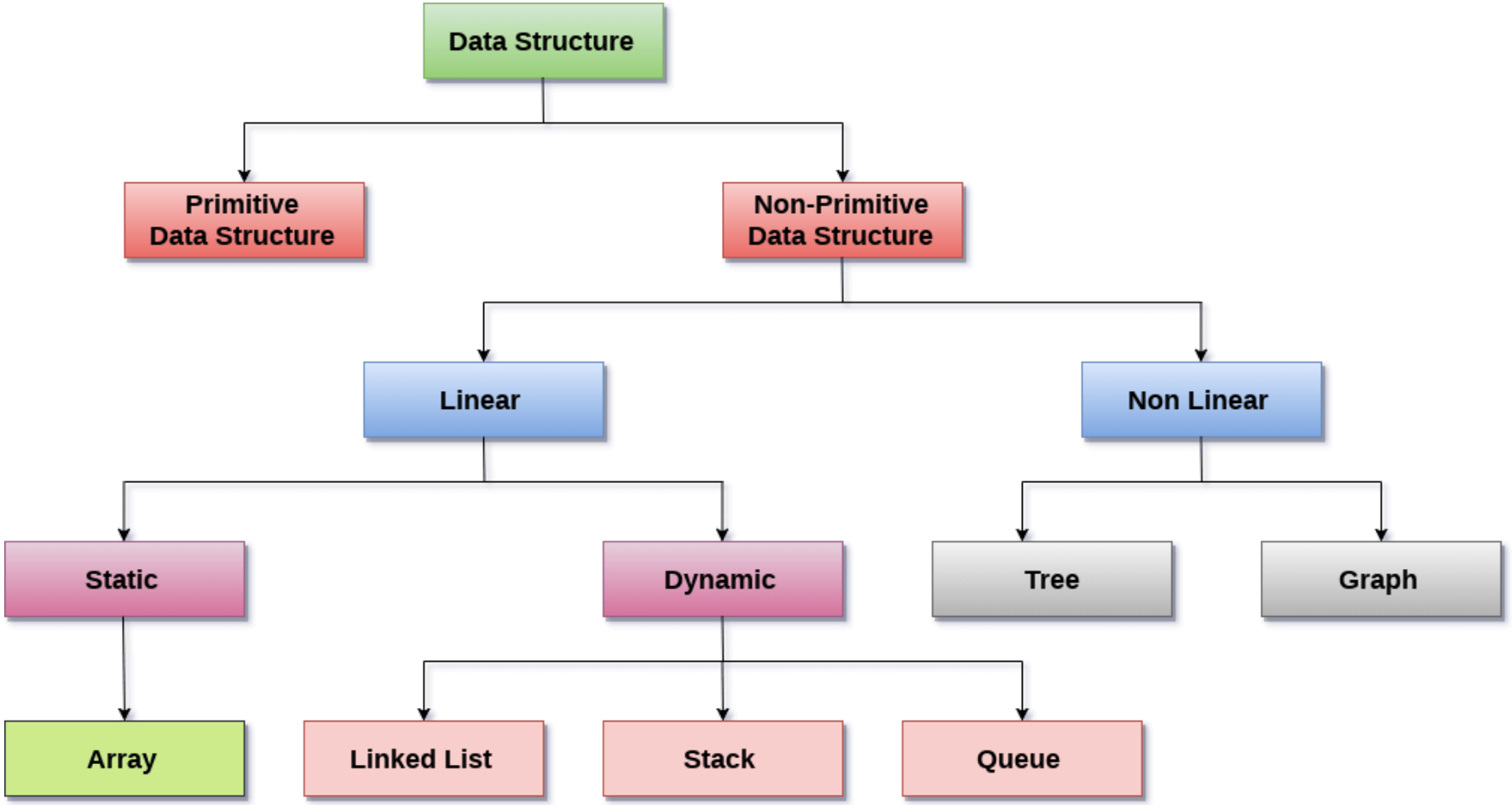
Lecture 09 – Linked List

Friday, November 10, 2023

Data Structure

fti.unai.edu

Data Structures Classifications



Linked List as Abstract Data Types

- **Linked List** adalah serangkaian elemen data linier (nodes). Urutan data ditentukan menggunakan pointer.
- Sebuah linked list yang hanya memiliki 1 penghubung ke node lain disebut sebagai **single (singly) linked list**.
- Setiap node terbagi atas dua bagian berbeda:
  - Bagian pertama menyimpan informasi mengenai elemen atau node
  - Bagian kedua berisi alamat dari node berikutnya (link/next pointer field)

Linked List Representation

- **Linked List** dapat dikatakan sebagai serangkaian node di mana setiap node memiliki setidaknya satu pointer tunggal ke node berikutnya. Dalam kasus node terakhir, **null pointer** digunakan untuk mewakili bahwa tidak akan ada lagi node dalam linked list.

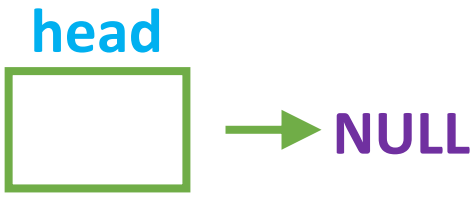
- Alamat dari node pertama disimpan dalam sebuah node yang dikenal dengan **HEAD**.

How Next Node is Referenced?

- Setiap node dalam single linked list terdiri atas:
  - Data item
  - Alamat dari node berikutnya
- Kedua elemen tersebut akan disimpan di dalam sebuah struct seperti di bawah ini:

```
struct node {
      int data;
      struct node *next;
}
```

How Next Node is Referenced?




```
struct node {
        int data;
        node *next;
}
node *head; //pointer untuk menyimpan alamat dari node pertama dalam list
head = NULL;
```

How Next Node is Referenced?


```
struct node {
      int data;
      node *next;
}
node *head; //pointer untuk menyimpan alamat dari node pertama dalam list
head = NULL;
node *nodeBaru = new node();
```

How Next Node is Referenced?

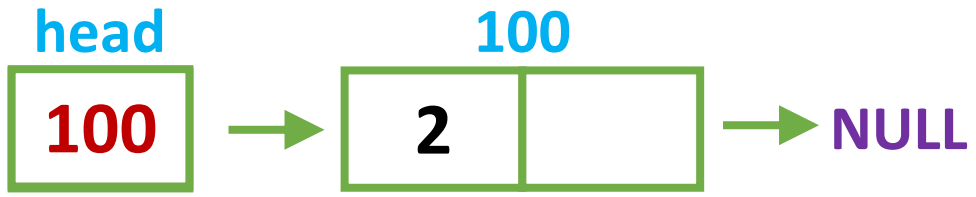

```
struct node {
      int data;
      node *next;
}
node *head; //pointer untuk menyimpan alamat dari node pertama dalam list
head = NULL;
node *nodeBaru = new node();
nodeBaru -> data = 2;
```

How Next Node is Referenced?


```
struct node {
      int data;
      node *next;
}
node *head;  //pointer untuk menyimpan alamat dari node pertama dalam list
head = NULL;
node *nodeBaru = new node();
nodeBaru -> data = 2;
nodeBaru -> next = NULL;
```

How Next Node is Referenced?

```
struct node {
      int data;
      node *next;
}
node *head;  //pointer untuk menyimpan alamat dari node pertama dalam list
head = NULL;
node *nodeBaru = new node();
nodeBaru -> data = 2;
nodeBaru -> next = NULL;
head = nodeBaru;
```



Linked List Operations

Terdapat setidaknya 3 operasi yang dapat dilakukan dalam linked list:

- **Insert** – menambahkan elemen ke dalam list
- **Delete** – menghapus elemen dari list
- **Traverse** – menelusuri list

Insertion in Linked List – Add to the Beginning

Proses penambahan elemen dalam linked list dapat dilakukan dengan 3 skenario:

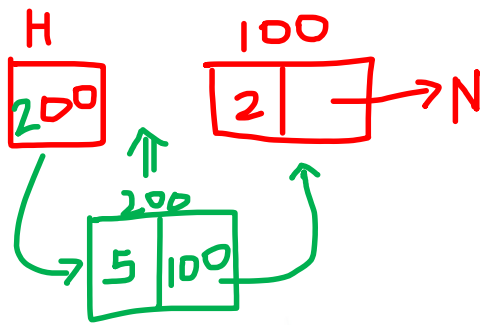
1) Add to the beginning

- Alokasikan memori untuk node yang baru
- Simpan data
- Ubah next dari node baru untuk sesuai dengan nilai head
- Ubah head untuk menunjuk ke node yang baru saja ditambahkan

```
node *nodeBaru = new node();
nodeBaru -> data = data;
nodeBaru -> next = head;
head = nodeBaru;
```

Insertion in Linked List – **Add to the Beginning**

```
void insertDiAwal(int bilangan){
   ✓ Node *nodeBaru = new Node();
    ✓ nodeBaru -> data = bilangan;
   ✓ nodeBaru -> next = head;
    ✓head = nodeBaru;
}
```



Insertion in Linked List – **Add to the End**

Proses penambahan elemen dalam linked list dapat dilakukan dengan 3 skenario:

2) Add to the end

- Alokasikan memori untuk node yang baru
- Simpan data
- Telusuri list sampai node terakhir
- Ubah next dari node terakhir untuk menunjuk ke node yang baru ditambahkan

```
node *nodeBaru = new node();
nodeBaru -> data = data;
nodeBaru -> next = NULL;
```

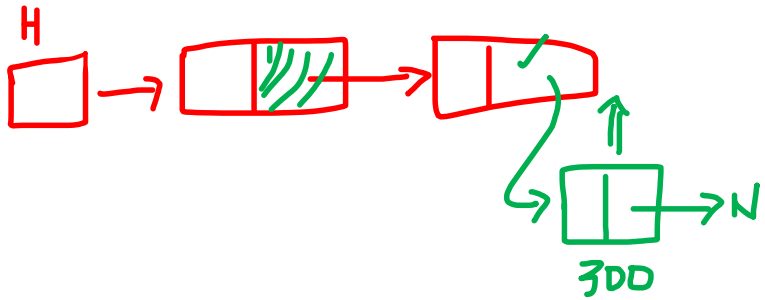
```
node *temp = new node();
while(temp->next != NULL){
      temp = temp->next;
}
temp->next = nodeBaru;
```

Insertion in Linked List – **Add to the End**

```
void insertDiAkhir(int bilangan){
    Node *nodeBaru = new Node();
    Node *temp = head;
    nodeBaru -> data = bilangan;
    nodeBaru -> next = NULL;

    if(head==NULL){
        head = nodeBaru;
        return;
    }

    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = nodeBaru;
    return;
}
```



Insertion in Linked List – **Add to n-th position**

Proses penambahan elemen dalam linked list dapat dilakukan dengan 3 skenario:

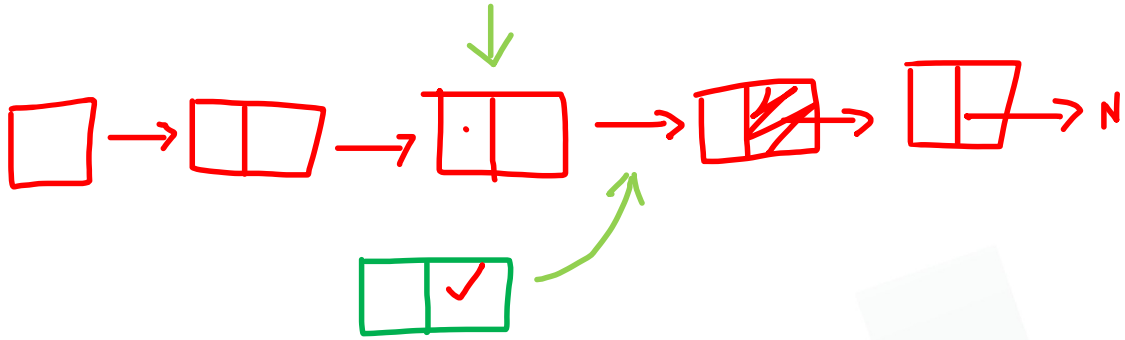
3) Add to the Middle

- Alokasikan memori untuk node yang baru
- Simpan data
- Telusuri list sampai posisi dimana node baru akan ditambahkan
- Ubah pointer berikutnya untuk memasukkan simpul baru di antaranya

Insertion in Linked List – **Add to n-th position**

```
void insertDiTengah(int bilangan, int pos){
    Node *nodeBaru = new Node();
    nodeBaru->data = bilangan;
    nodeBaru->next = NULL;

    if(pos==1){
        nodeBaru->next = head;
        head = nodeBaru;
        return;
    }
    Node *temp = head;
    for(int i=0;i<pos-2;i++){
        temp=temp->next;
    }
    nodeBaru->next = temp->next;
    temp->next = nodeBaru;
}
```



FAKULTAS TEKNOLOGI INFORMASI
Universitas Advent Indonesia

Deletion in Linked List

Proses penghapusan elemen dari linked list dapat dilakukan dengan 3 skenario berikut:

1) Delete from beginning

- Atur head untuk menunjuk ke node yang kedua

2) Delete from end

- Telusuri list sampai ke elemen kedua terakhir
- Ubah pointer next dari elemen tersebut menjadi NULL

3) Delete from middle (n-th position)

- Telusuri list sampai ke elemen sebelum elemen yang akan dihapus
- Ubah pointer next untuk dikeluarkan dari list

Deletion in Linked List

```
Node *temp = head;
if(pos==1){
    head = temp->next;
    delete temp;
    return;
}
for(int i=0;i<pos-2;i++){
    temp=temp->next;
}
Node *temp1 = temp->next;
temp->next = temp1->next;
delete temp1;
```

FAKULTAS TEKNOLOGI INFORMASI
Universitas Advent Indonesia

Traversing a Linked List

- Untuk menelusuri linked list atau menampilkan isi dari linked list dapat dilakukan dengan membuat suatu variabel/node sementara (nodeBaru) yang nantinya akan berpindah dari satu node ke node berikutnya kemudian tampilkan isinya.
- Ketika node nodeBaru = NULL itu berarti kita sudah mencapai bagian akhir dari linked list.

```
struct node *nodeBaru = head;
cout << “Elemen dalam list adalah \n”;
while(nodeBaru != NULL){
      cout << nodeBaru -> data << “ ”;
      nodeBaru = nodeBaru -> next;
}
```

Class Exercise

- Buatlah sebuah program yang berisi Menu Lengkap operasi pada Linked List. Tambahkan satu fungsi untuk menghapus berdasarkan nilai elemen yang terdapat dalam list.

- **Lihat contoh output..**