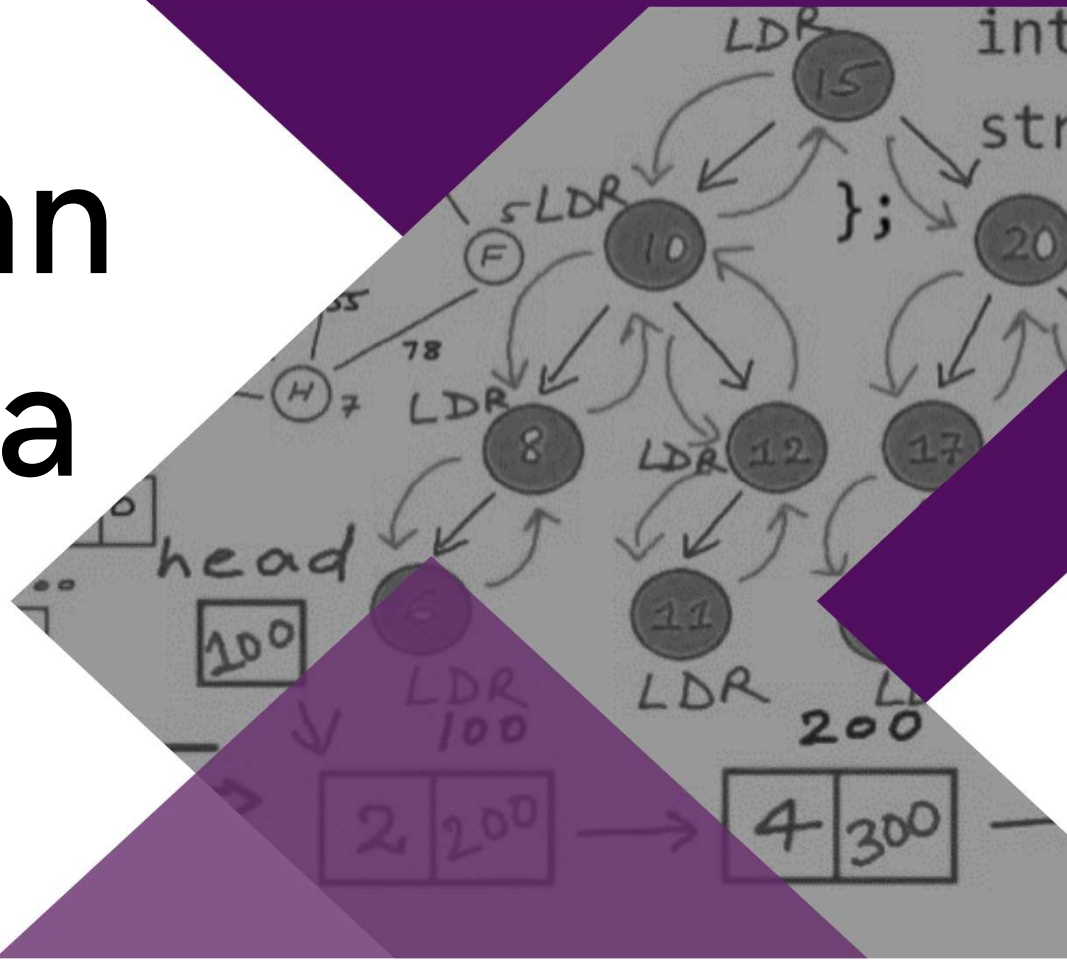
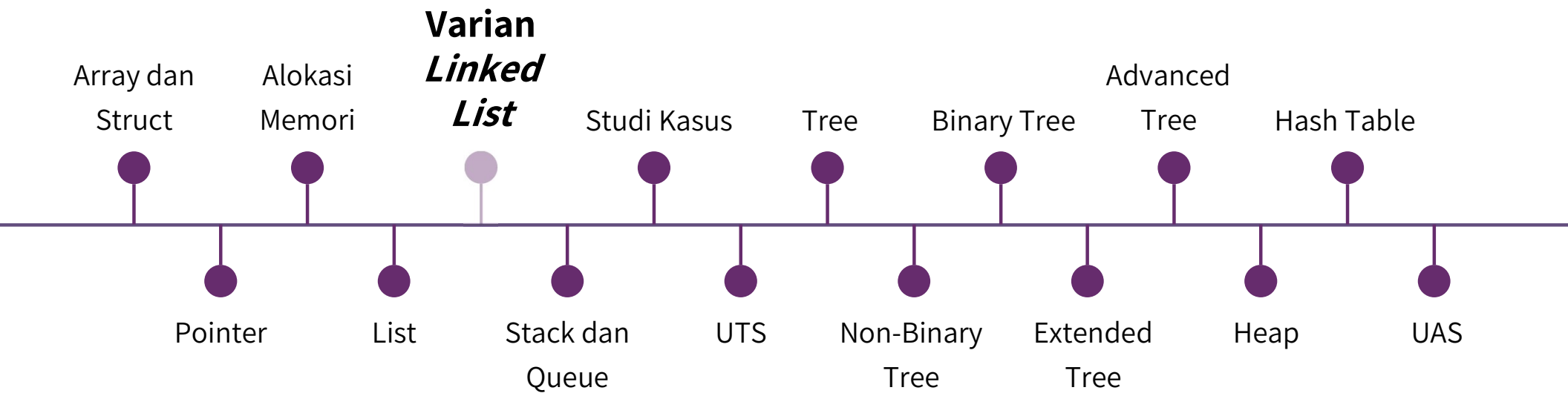
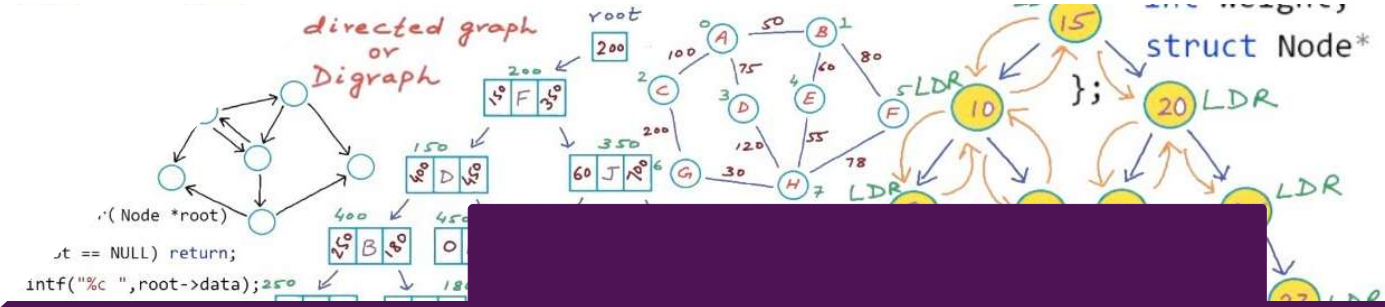
Algoritma dan Struktur Data



Pekan 5

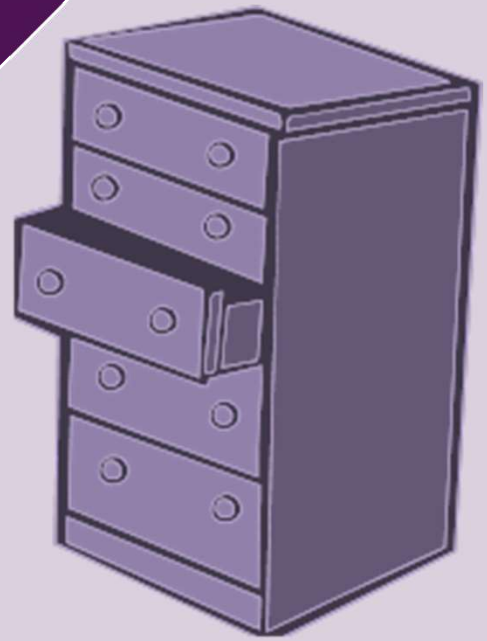

Array dan Struct
Alokasi Memori
Varian *Linked List*
Studi Kasus
Tree
Binary Tree
Advanced Tree
Hash Table
Pointer
List
Stack dan Queue
UTS
Non-Binary Tree
Extended Tree
Heap
UAS

Tujuan

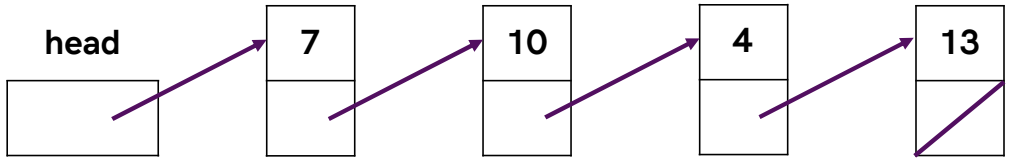


1	Mahasiswa memahami perbedaan *single linked-list*, *doubly linked-list*, dan *circular linked-list*.
2	Mahasiswa mampu mengimplementasikan *single linked-list, doubly linked-list*, dan *circular linked-list*

Circular Linked List



Rationale

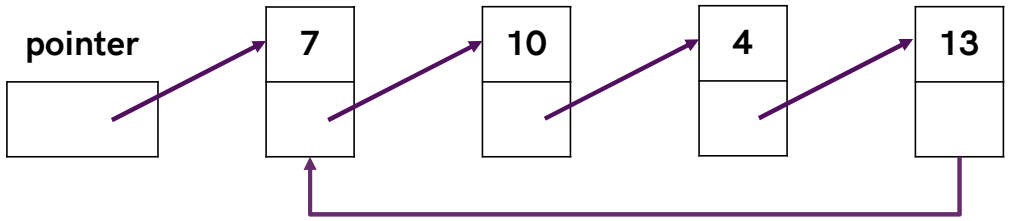


Problem: Given a pointer to a node anywhere in the list, we can access all the nodes that follow but none of the nodes that precede it.

We must always have a pointer to the beginning of the list to be able to access all the nodes in the list.

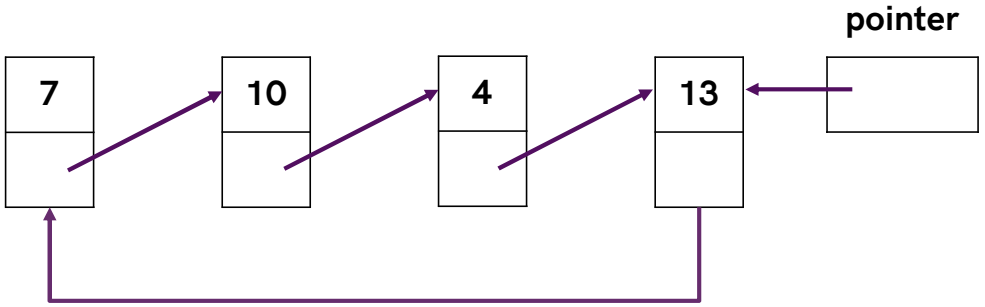
Circular Linked List

A linear data structure in which every node has a successor; the last element is succeeded by the first element.

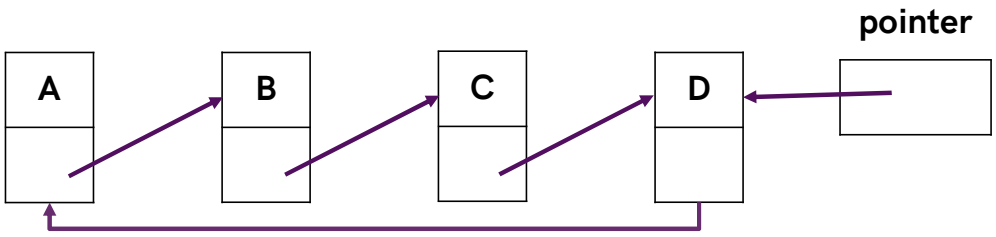


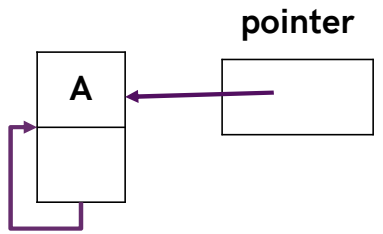
Implementation changes:

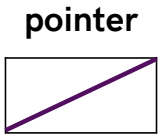
- Traversing list



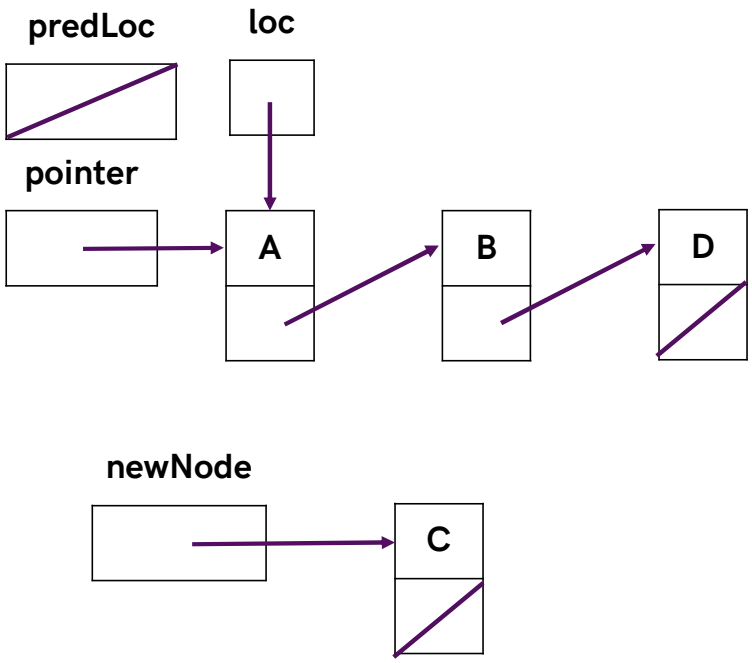
Example 1 - IsEmpty



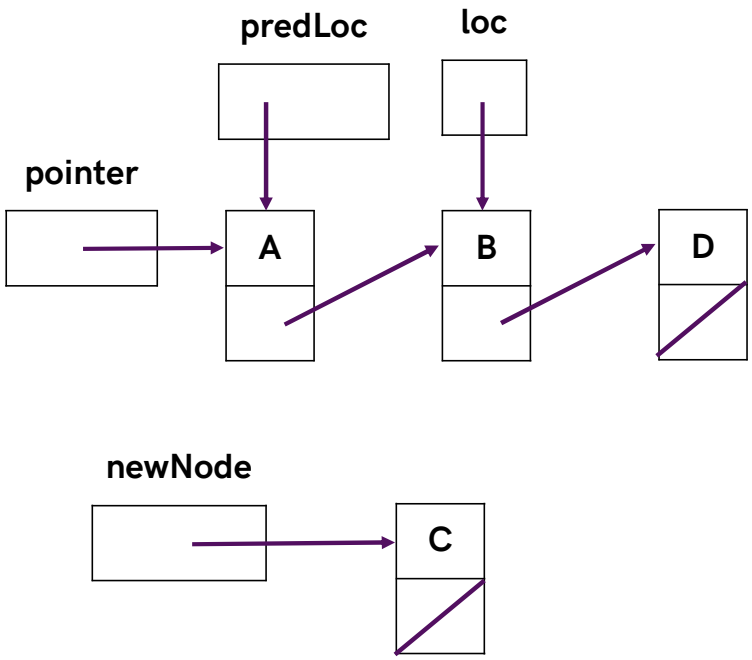




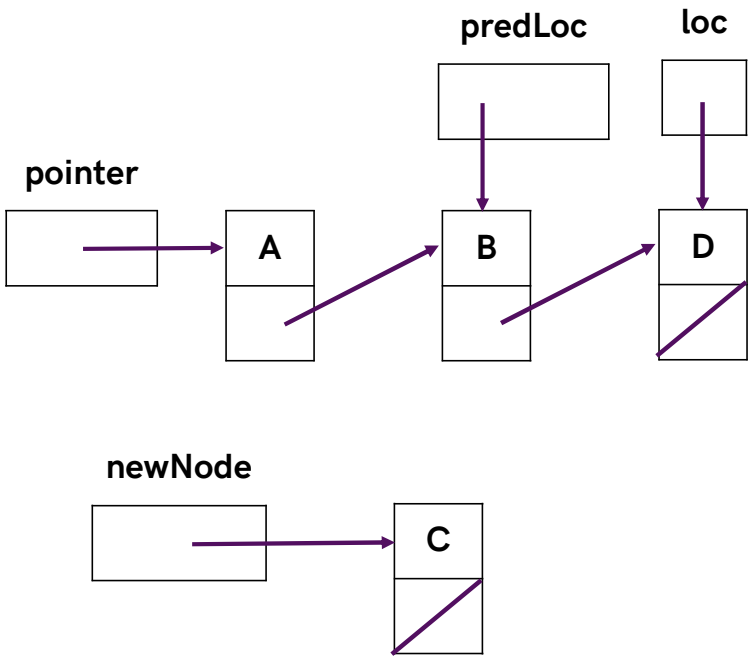
Example 2 – Traversing List



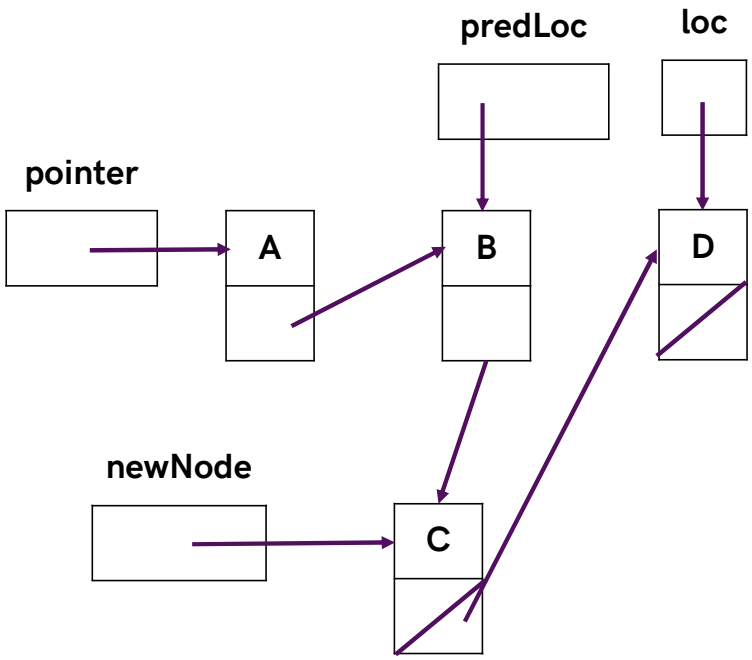
Example 2 – Traversing List



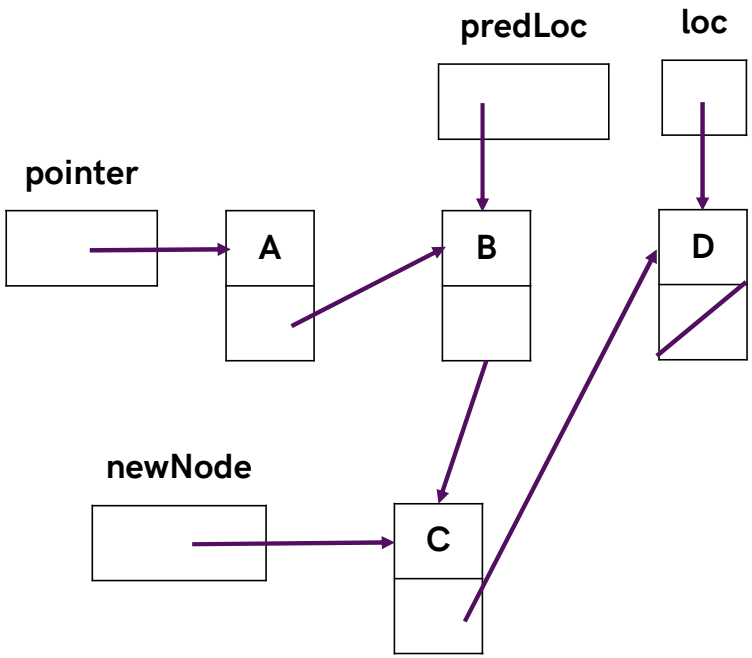
Example 2 – Traversing List

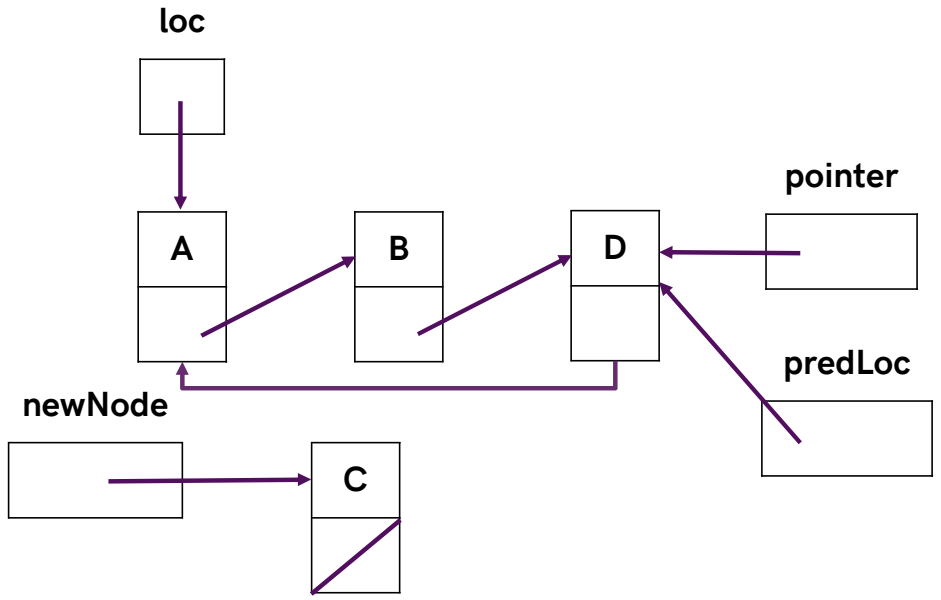


Example 2 – Traversing List

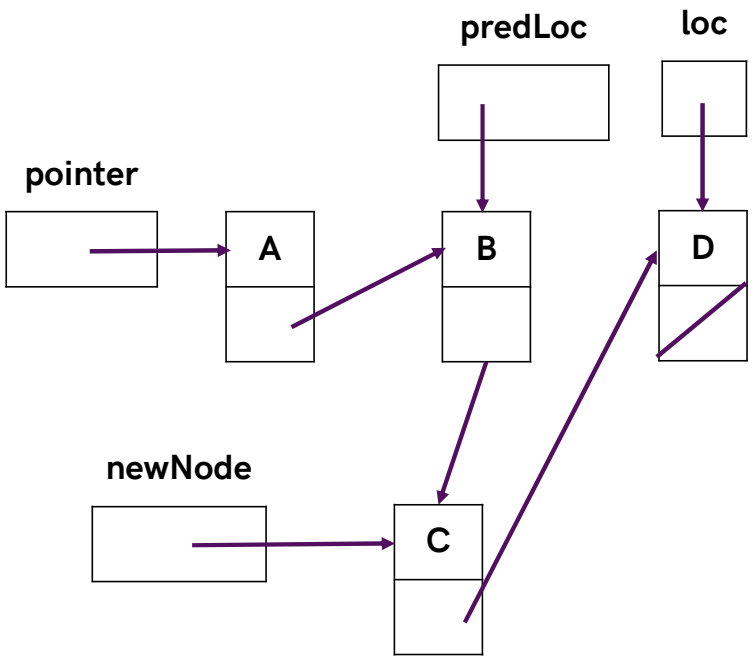


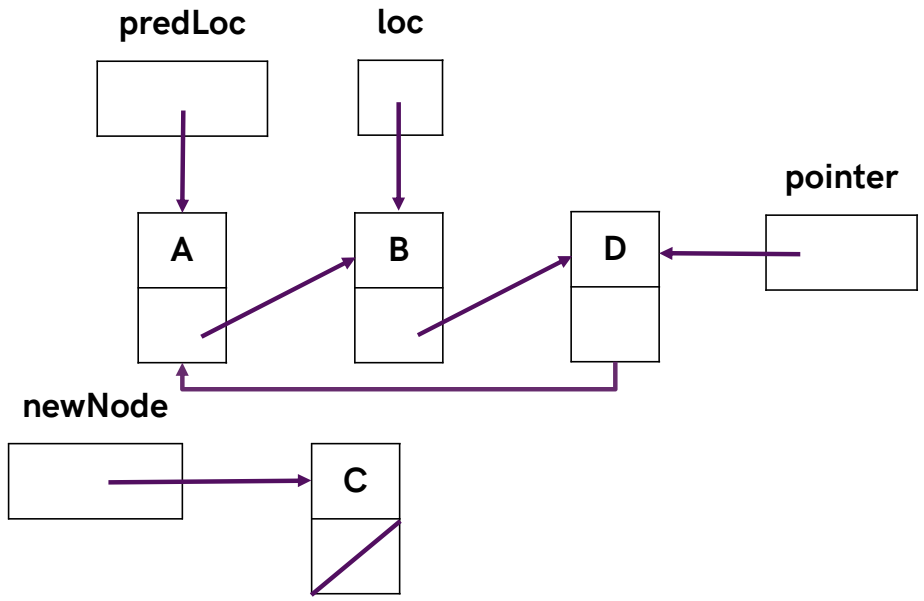
Example 2 – Traversing List



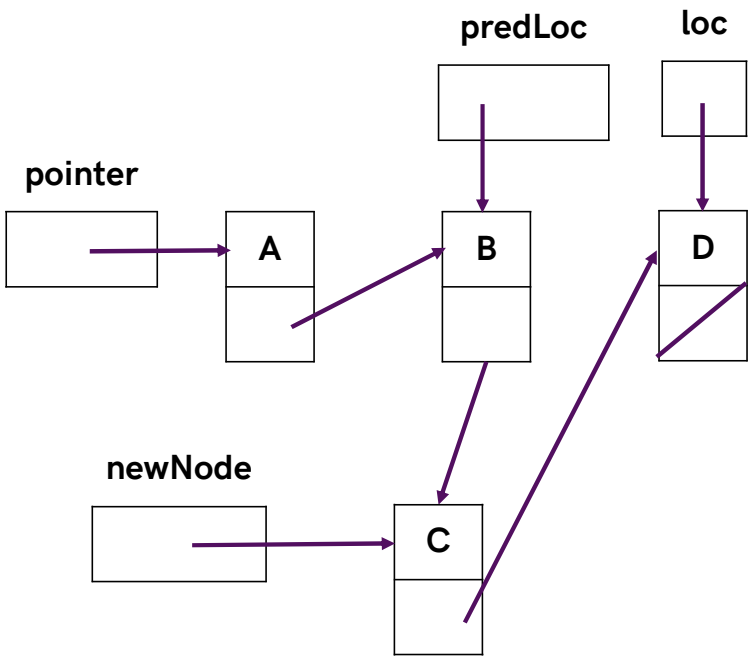


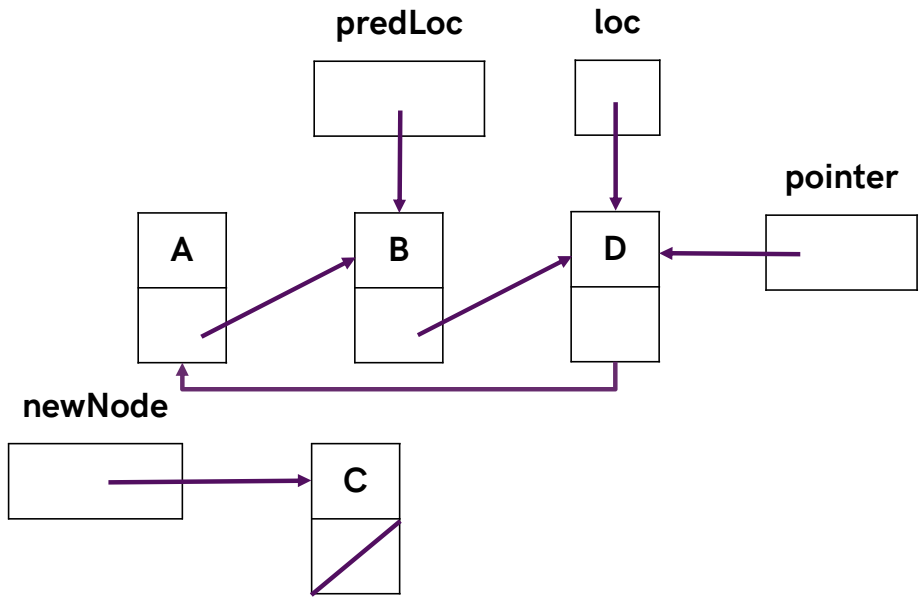
Example 2 – Traversing List



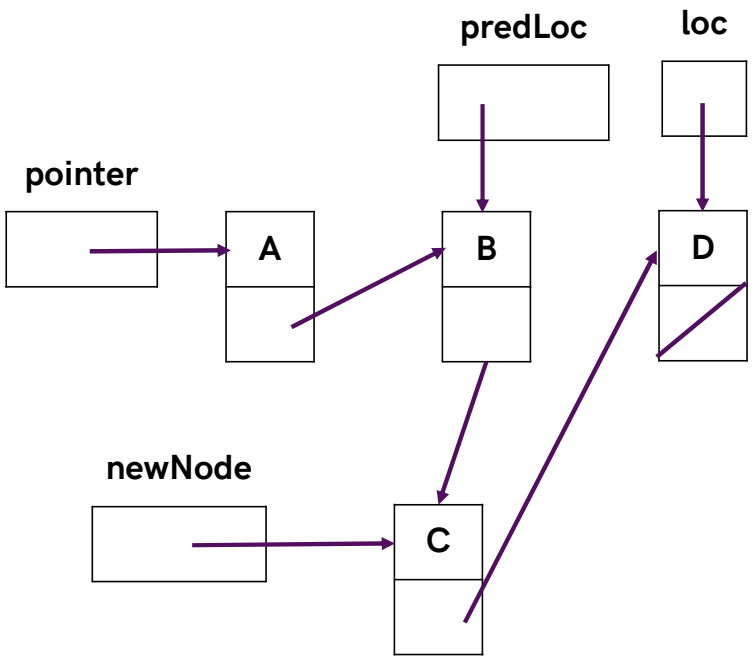


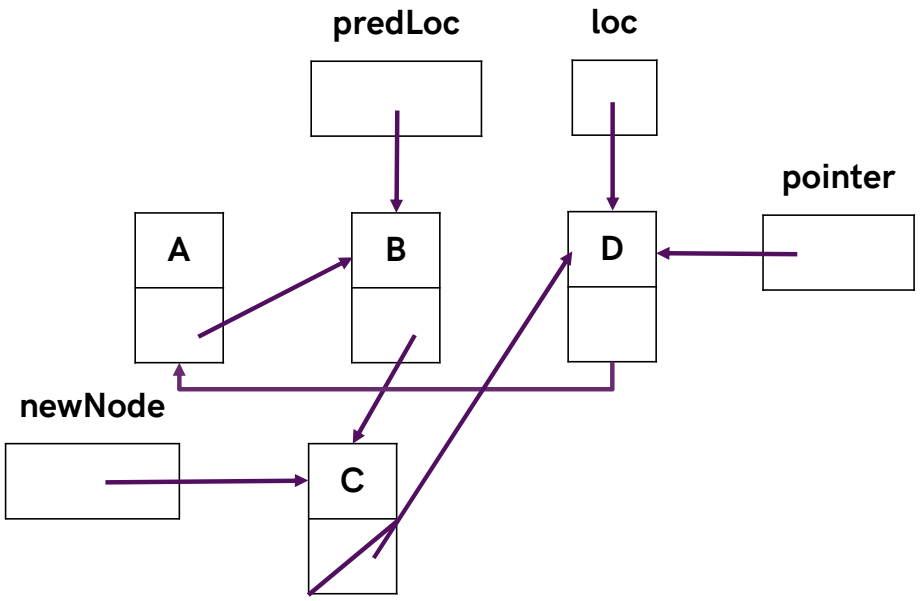
Example 2 – Traversing List





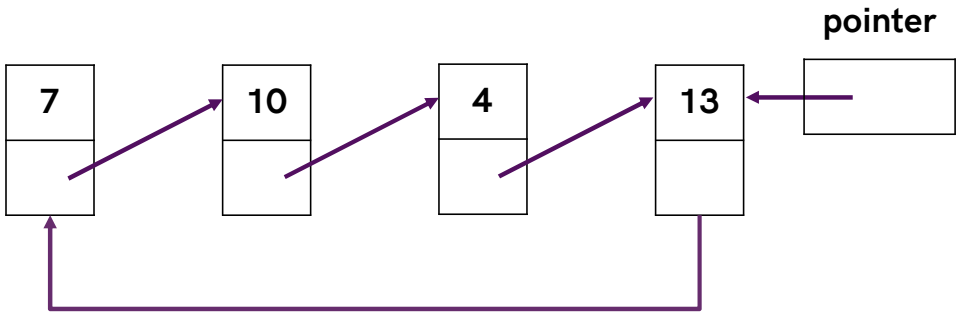
Example 2 – Traversing List





Doubly Linked List

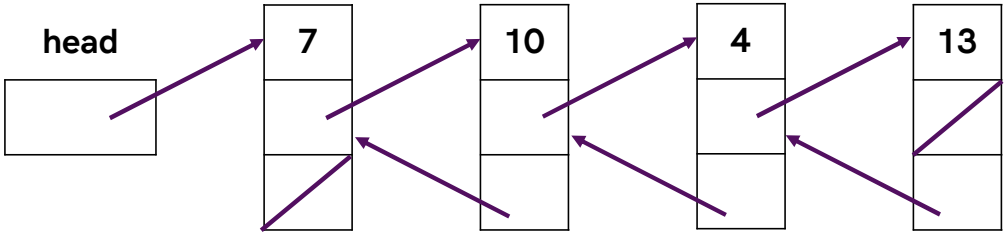
Rationale



Problem: Difficult to traverse the list in reverse

Doubly Linked List

A linear data structure in which every node is linked to both its successor and its predecessor



Node contains:

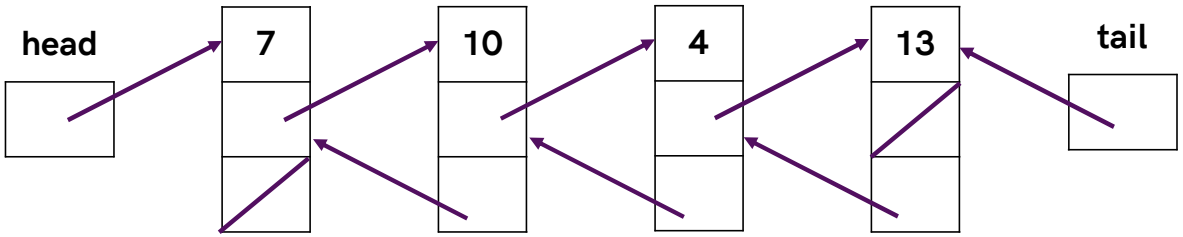
- key / data
- Next pointer
- Previous pointer

Implementation Changes

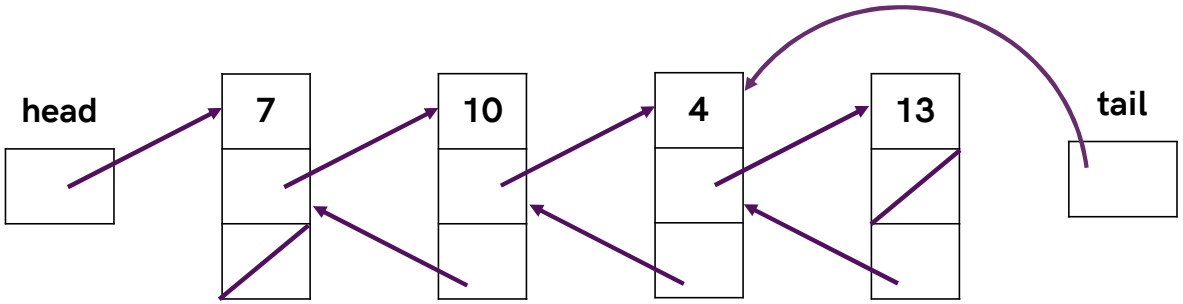
- Declaration
- Traversing list

```
typedef struct node {
    int x;
    node *next_p;
    node *prev_p;
} node_t;
```

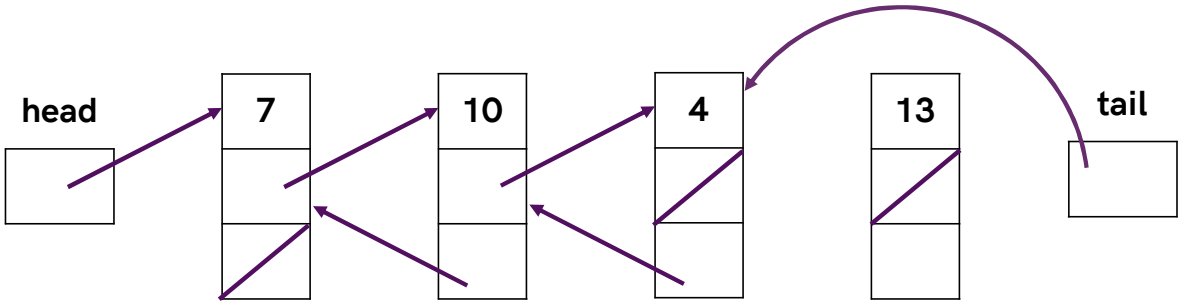
Example - PopBack



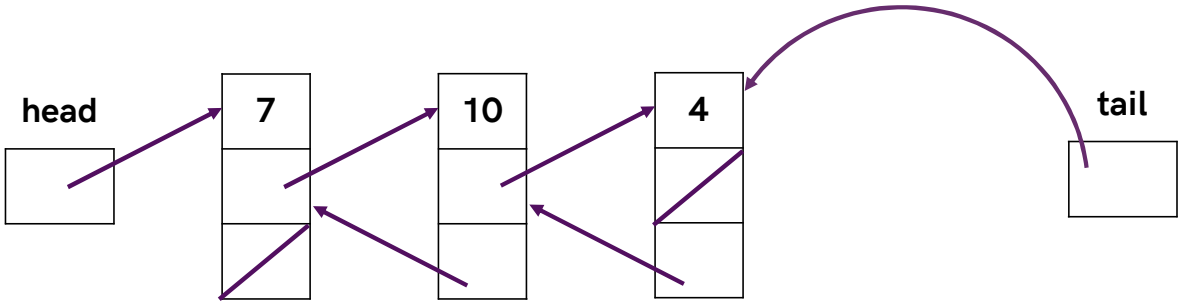
Example - PopBack



Example - PopBack



Example - PopBack



O(1)

Implementation

PushBack(*key*)

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

Implementation

PopBack()

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

Implementation

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

Implementation

```
AddBefore(node, key)
node2 ←new node
node2.key ← key
node2.next ← node
node2.prev ← node.prev
node.prev ← node2
if node2.prev ≠nil:
   node2.prev.next ← node2
if head = node:
   head ← node2
```

Kompleksitas

Singly Linked List	No Tail	With Tail
PushFront(key)	O(1)	
TopFront()	O(1)	
PopFront()	O(1)	
PushBack(key)	O(n)	O(1)
TopBack()	O(n)	O(1)
PopBack()	O(n)	
Find(key)	O(n)	
Erase(key)	O(n)	
Empty()	O(1)	
AddBefore(Node, Key)	O(n)	
AddAfter(Node, Key)	O(1)	

Doubly Linked List	No Tail	With Tail
PushFront(key)	O(1)	
TopFront()	O(1)	
PopFront()	O(1)	
PushBack(key)	O(n)	O(1)
TopBack()	O(n)	O(1)
PopBack()	O(1)	
Find(key)	O(n)	
Erase(key)	O(n)	
Empty()	O(1)	
AddBefore(Node, Key)	O(1)	
AddAfter(Node, Key)	O(1)	