Data Structures & Algorithms

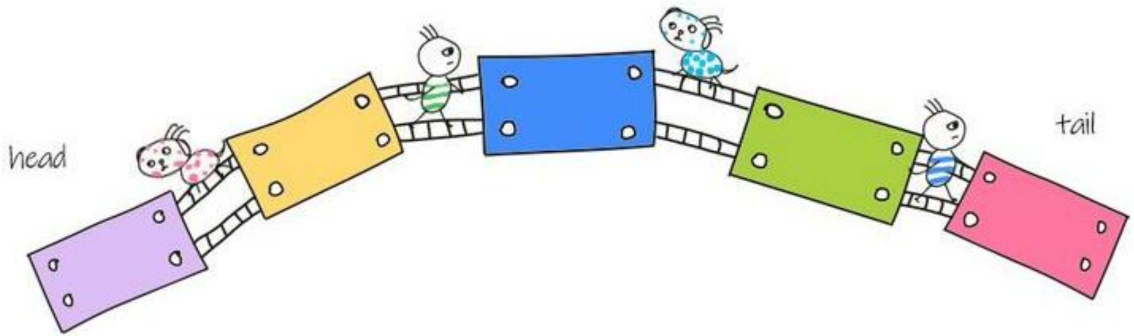
05: Linked List; Part - II

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

Linked List



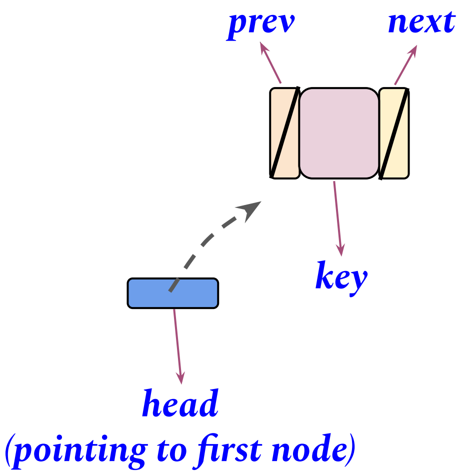
Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List

Object / Node



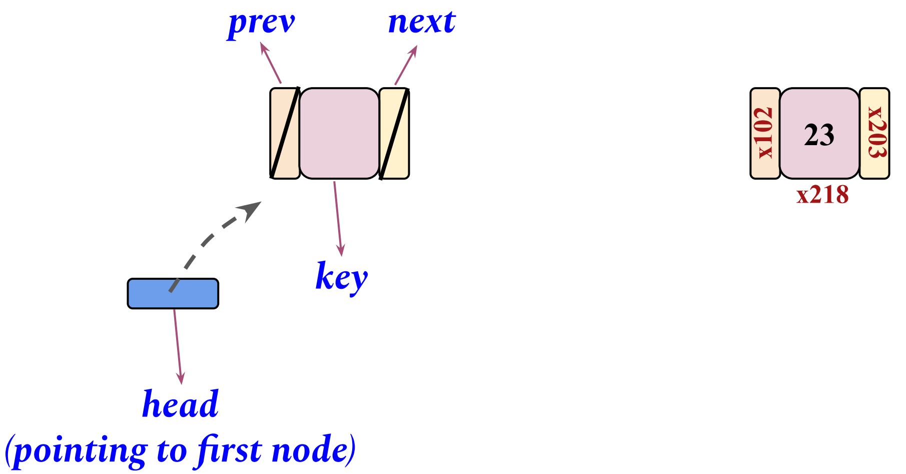
Linked List

Linked List: A linked list is a fundamental data structure that stores elements in a **linear order**, but unlike arrays, **not necessarily in contiguous memory locations**.

- Order of the data stored in a linked list is determined by the pointer in each object/node.

Doubly Linked List

Object / Node

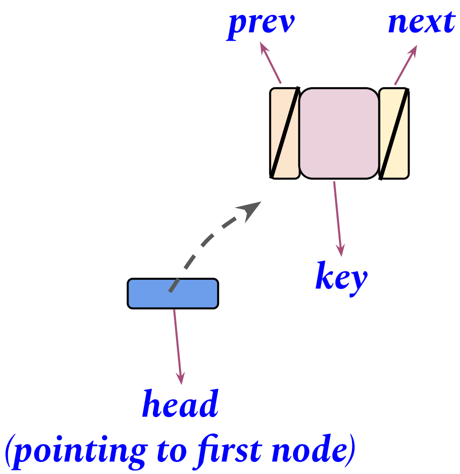


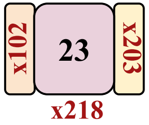
Linked List

Doubly Linked List

In a doubly linked list, each element x has the following attributes:

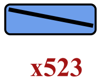
- ***x.key***
- ***x.next***: the successor of x, NIL if x has no **successor** so that it's the tail
- ***x.prev***: the predecessor of x, NIL if x has no **predecessor** so that it's the head
- ***L.head*** points to the first element of the list, NIL if the list is empty.



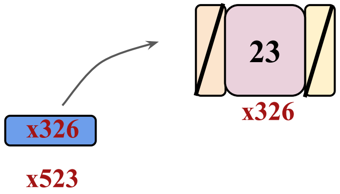


Linked List

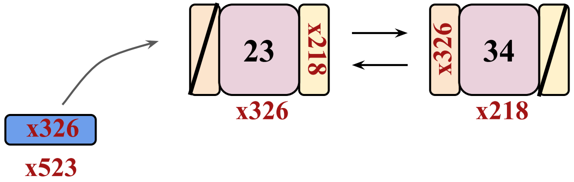
Empty list



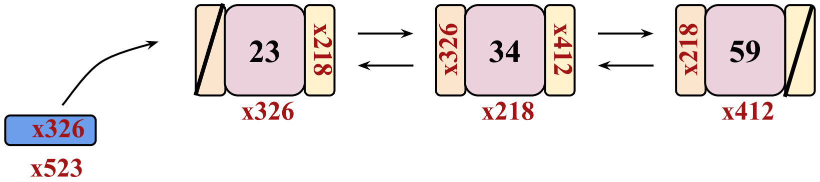
One element list



Two elements list

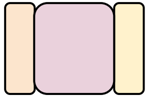


Three elements list

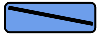


Linked List

```
struct node
{
 int key;
 struct node *prev;
 struct node *next;
};
```



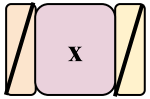
```
struct node *L_head = NULL; // Empty List
```



```
struct node *createNode(int x)
{
 struct node *newNode = (struct node *)malloc(1 * sizeof(struct node));

 newNode->key = x;
 newNode->prev = NULL;
 newNode->next = NULL;

 return newNode;
}
```

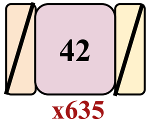


Inserting a new node

Linked List

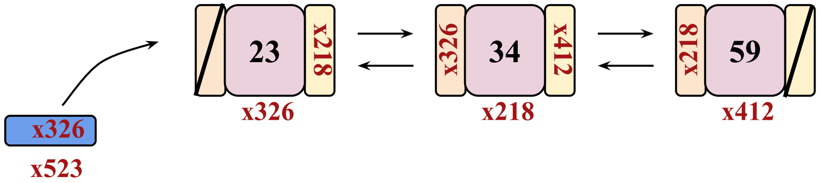
Request a new node:

```
struct node *newNode = createNode(42);
```



Find which node contains 34
⟹ i.e., the address of node

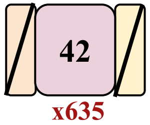
Insert 42 after 34



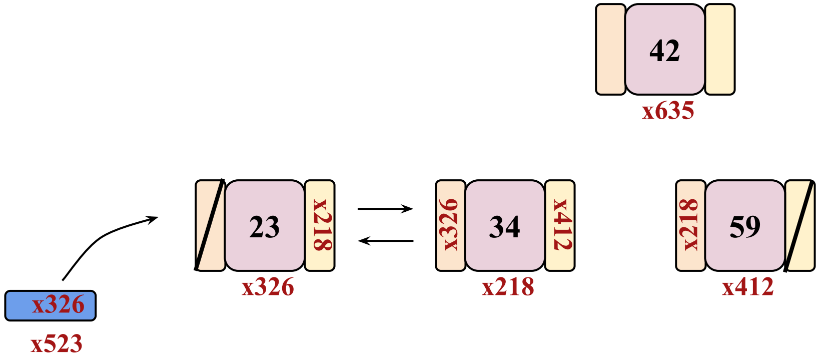
LINKED LIST

Request a new node:

```
struct node *newNode = createNode(42);
```



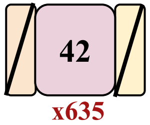
Insert 42 after 34



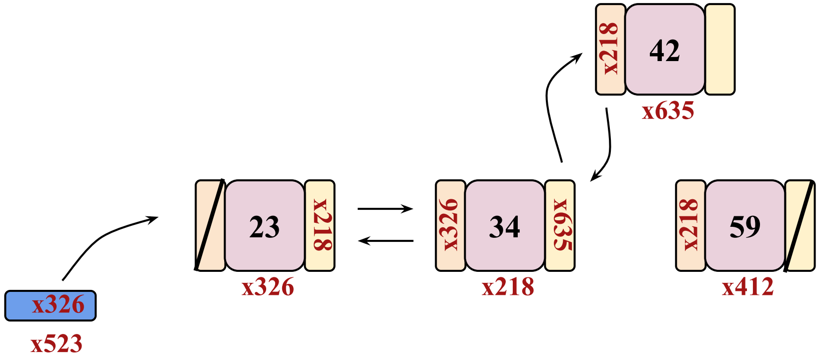
Linked List

Request a new node:

```
struct node *newNode = createNode(42);
```



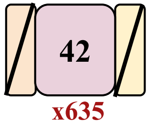
Insert 42 after 34



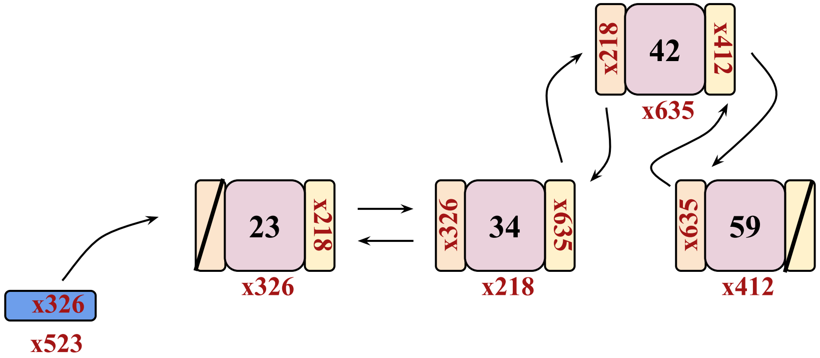
Linked List

Request a new node:

```
struct node *newNode = createNode(42);
```



Insert 42 after 34



Linked List

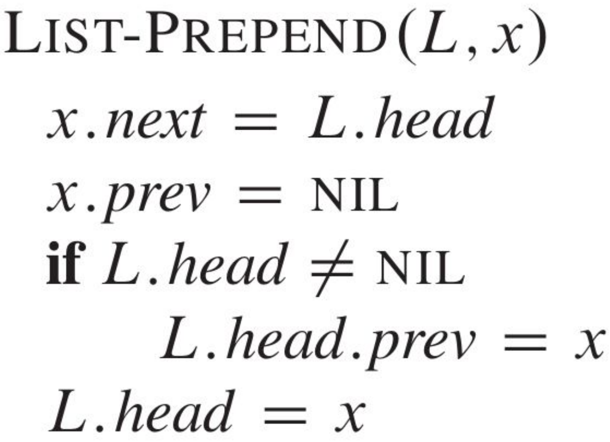
Few points to note while inserting:

- **List Prepend** → Add the node as first node.
    - The ***prev*** pointer of this node is always going to be **NULL**.

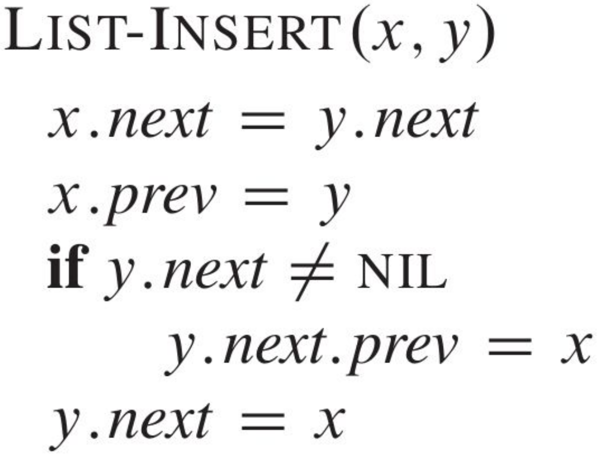
- **List Append** → Add the node as the last node.
    - The ***next*** pointer of this node is always going to be **NULL**.

- **List Insert** → Insert a node after a given node.
    - Locate the given node.
    - Update the ***prev*** and ***next*** pointers of the new node.

Linked List



```
LIST-PREPEND$(L, x)$
  $x.next = L.head$
  $x.prev =$ NIL
  if $L.head \neq$ NIL
      $L.head.prev = x$
  $L.head = x$
```

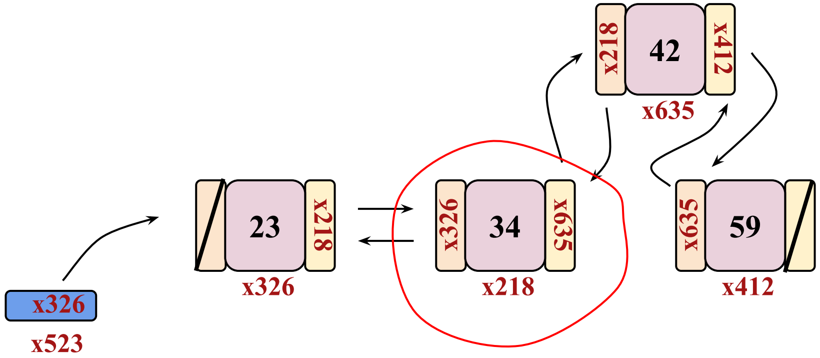


```
LIST-INSERT$(x, y)$
  $x.next = y.next$
  $x.prev = y$
  if $y.next \neq$ NIL
      $y.next.prev = x$
  $y.next = x$
```

Delete a node

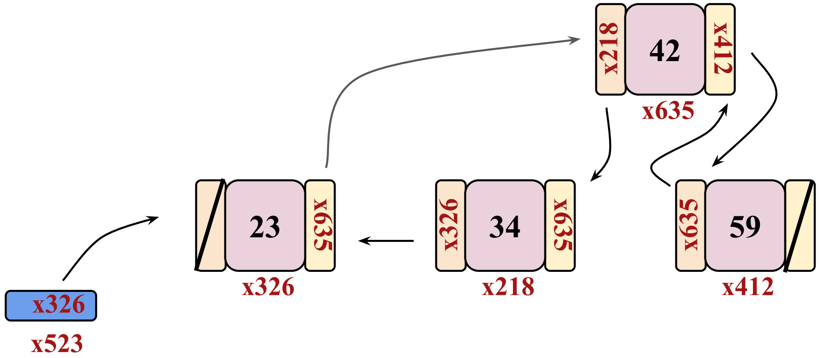
Linked List

Delete 34



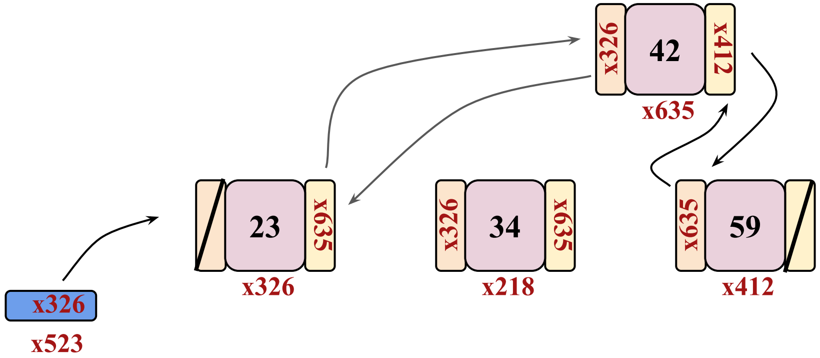
Linked List

Delete 34



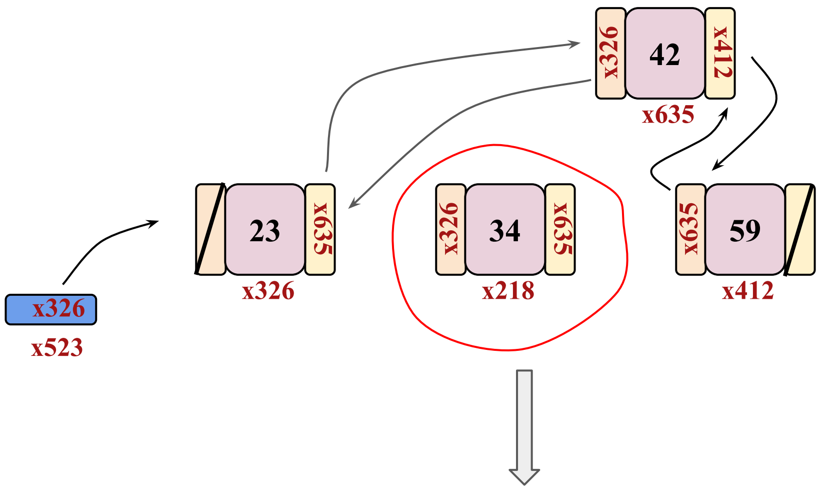
Linked List

Delete 34



Linked List

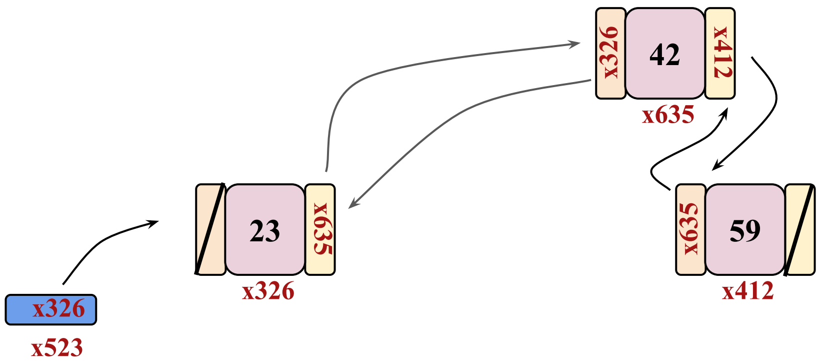
Delete 34



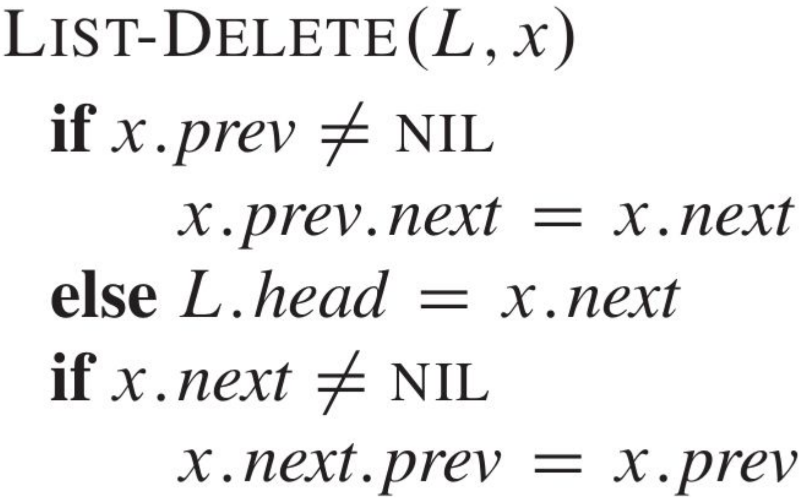
remove from the memory

Linked List

Delete 34



Linked List



```
LIST-DELETE($L, x$)
  if $x.prev \neq$ NIL
      $x.prev.next = x.next$
  else $L.head = x.next$
  if $x.next \neq$ NIL
      $x.next.prev = x.prev$
```

C Program for Doubly Linked List operations

(Pointer implementation)