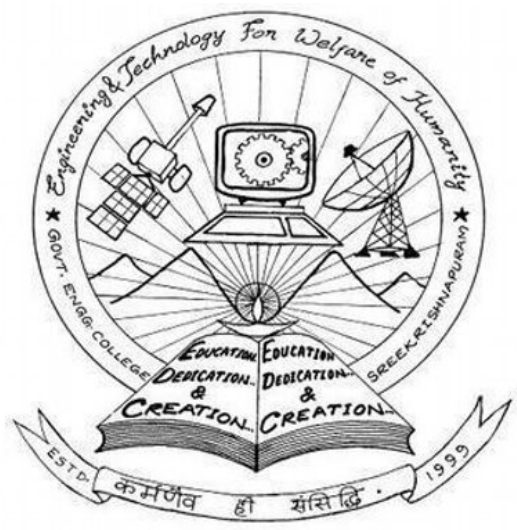
ITT201 Data Structures

Module 2 : Linked Lists



Anoop S K M
(https://sites.google.com/site/skmanoop)
Department of Information Technology
Govt. Engg. College, Sreekrishnapuram, Palakkad

Acknowledgements

- All the pictures are taken from the Internet using Google search.
- Wikipedia also referred.

Lecture 19

We Saw & Will See

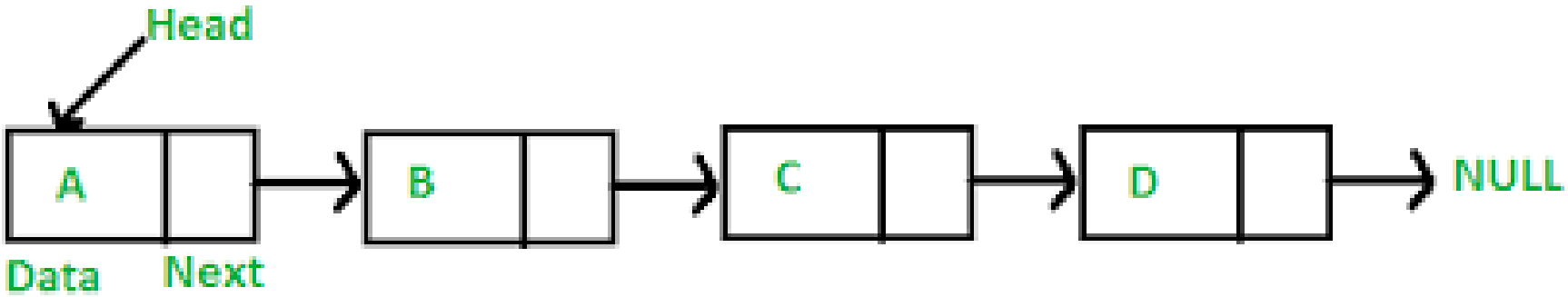
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Array
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications - Infix to Postfix, Reversal, Parenthesis Matching
  - Double Ended Queue, Priority Queue

Today We Will See...

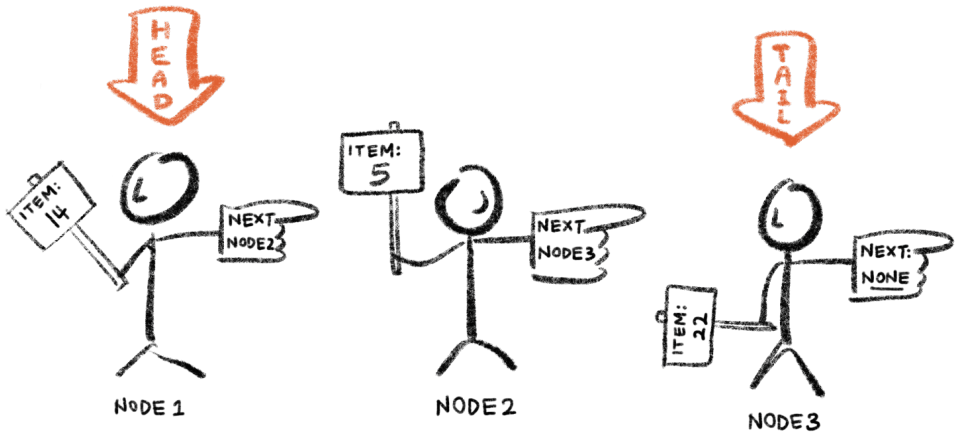
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List

Singly Linked List(SLL)



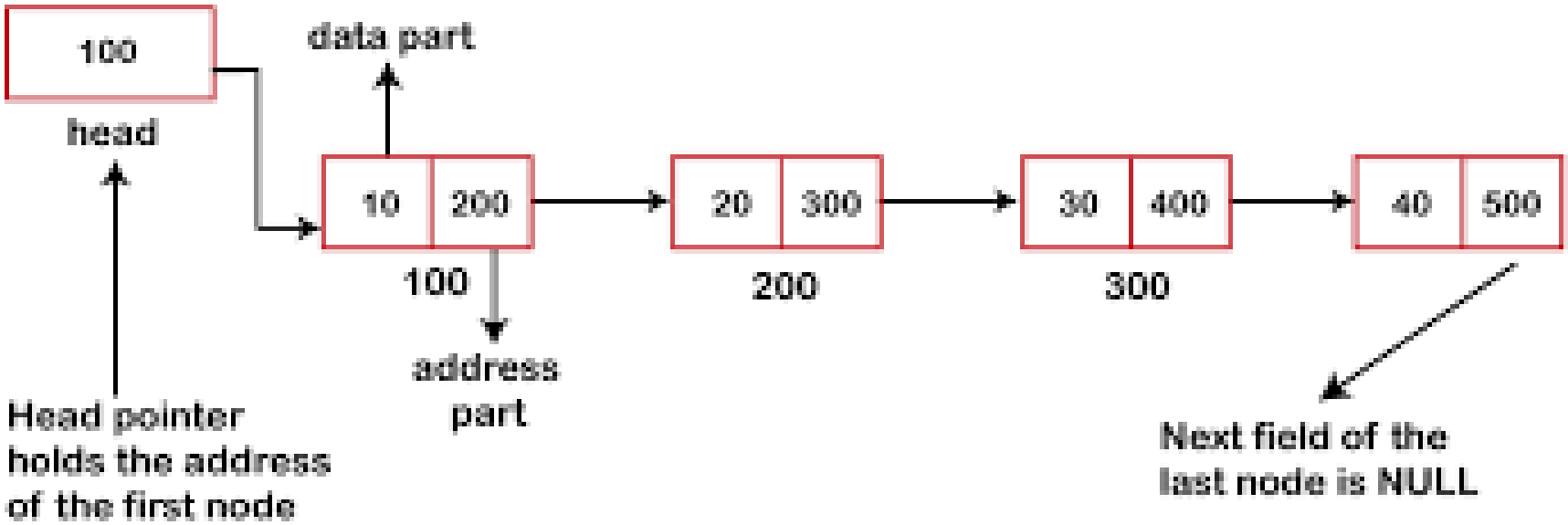
- head contains the address of the first node in the linked list
- every node contains a data part and a next part
- next part contains the address of the next node
- if next is NULL it is the end of the linked list

Single Linked List - Pictorial Representation

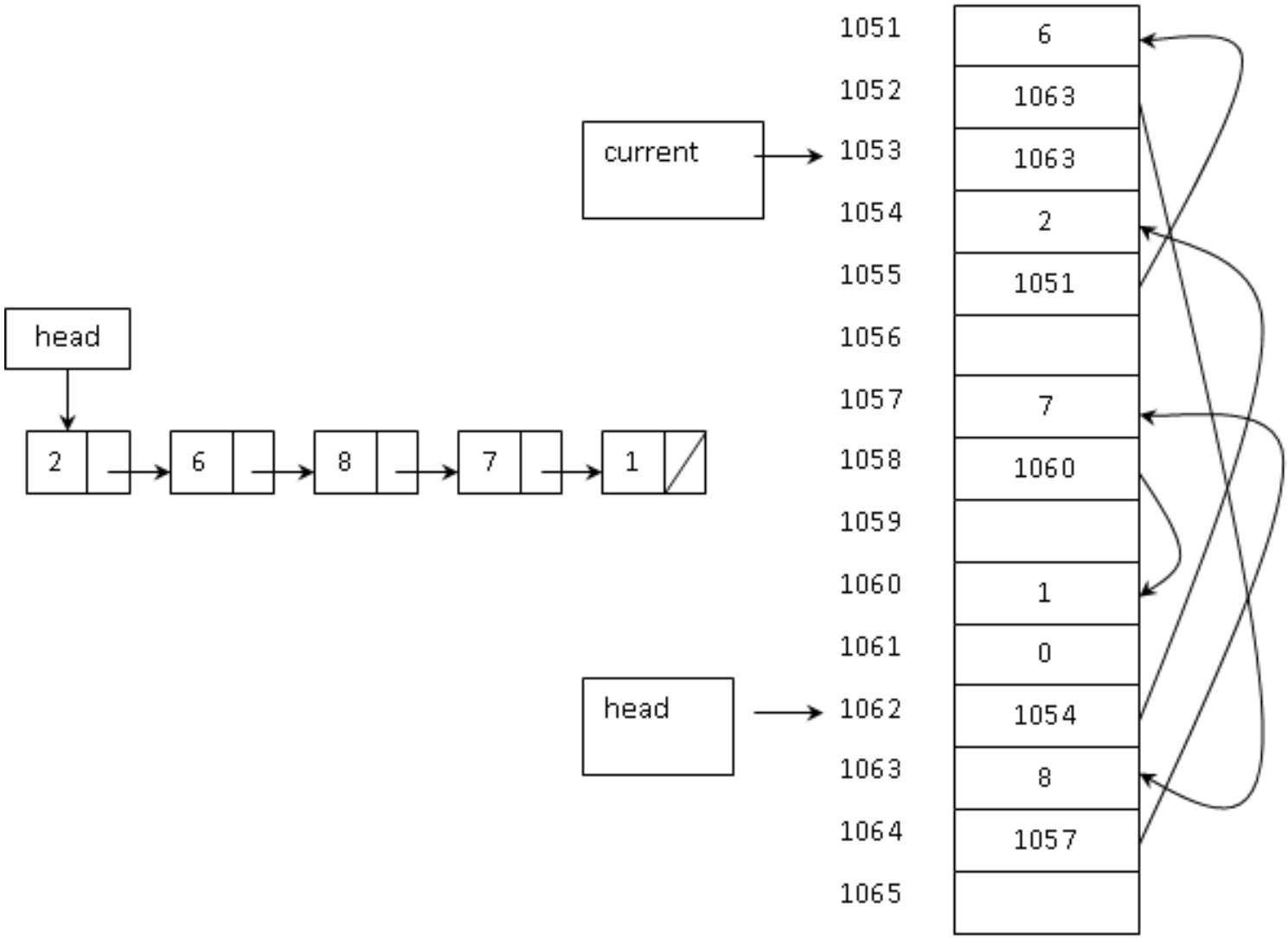


- In addition to head we may have a variable rear for storing the address of the last node.
- but generally we don't use it as we can identify the last node by looking at the next address present in it.

SLL - Example


100
head
data part
10
200
100
20
300
200
30
400
300
40
500
address
part
Head pointer
holds the address
of the first node
Next field of the
last node is NULL

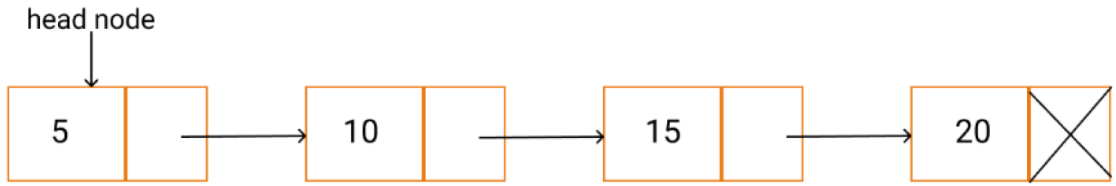
SLL implementation Example



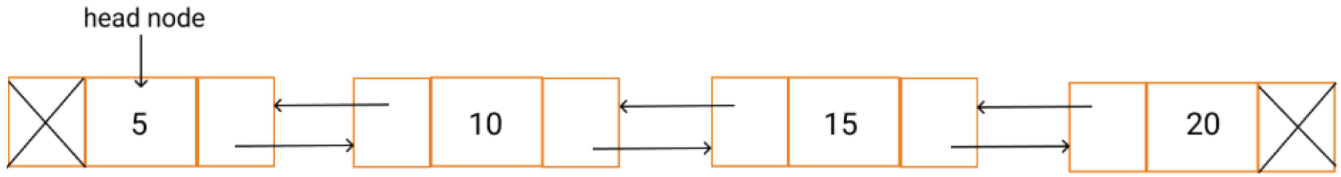
faceprep

Types of Linked List

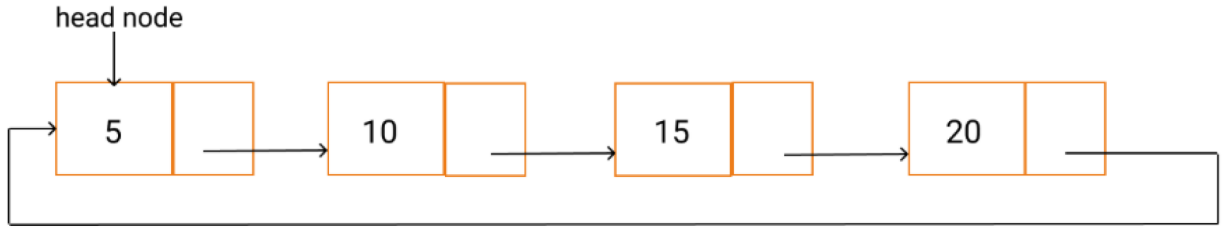
Singly Linked List



Doubly Linked List



Circular Linked List



Linked List Node

Defining a Node in C

```
struct node{
  int data;
  struct node* next;
};
struct node *head = NULL;
```

- similar to struct student we saw last semester.
- next is a pointer variable of node itself!
- head is a pointer used to store the starting address.
- initially head = NULL, i.e. linked list is empty.

Lecture 20

We Saw & Will See

Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Array
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications - Infix to Postfix, Reversal, Parenthesis Matching
  - Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List

Today We Will See...

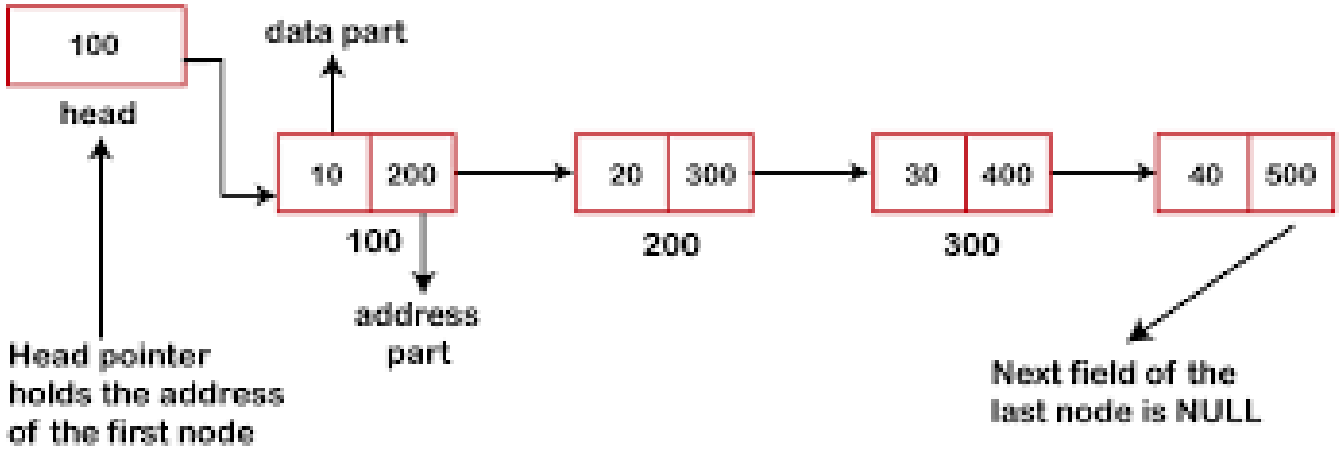
- Insertion at Front of SLL, Display the SLL

Linked List Node

Defining a Node in C

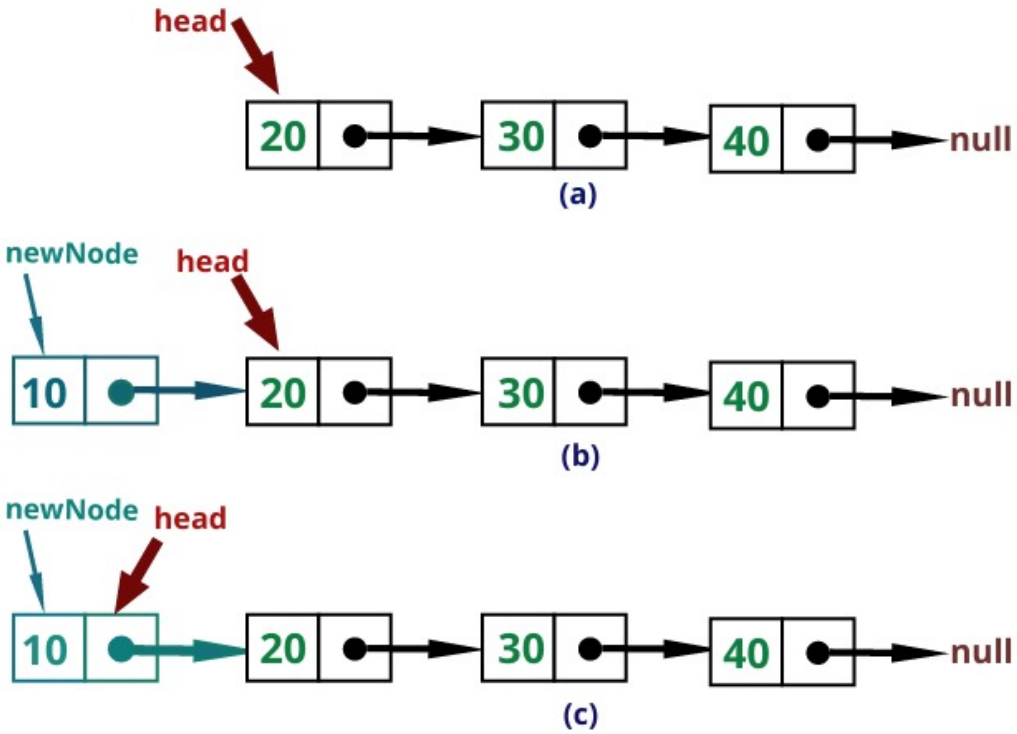
```
struct node{
  int data;
  struct node* next;
};
struct node *head = NULL;
```

Display the Linked List




```
p=head;
while(p!=NULL){
   printf("%d ",p->data);
   p=p->next;
}
printf("\n");
```

Insert a Node at Front -Idea



- Create a new node and put the value in the data part of it
- Make the next pointer points to the current head
- Change the head to this newly created node's address

Insert a Node at Front

```
int val;
struct node *newNode;
printf("Enter the value to be inserted\t");
scanf("%d",&val);
newNode = (struct node *)malloc(sizeof(struct node));
newNode->data = val;
newNode->next = head;
head = newNode;
```

Lecture 21

We Saw & Will See

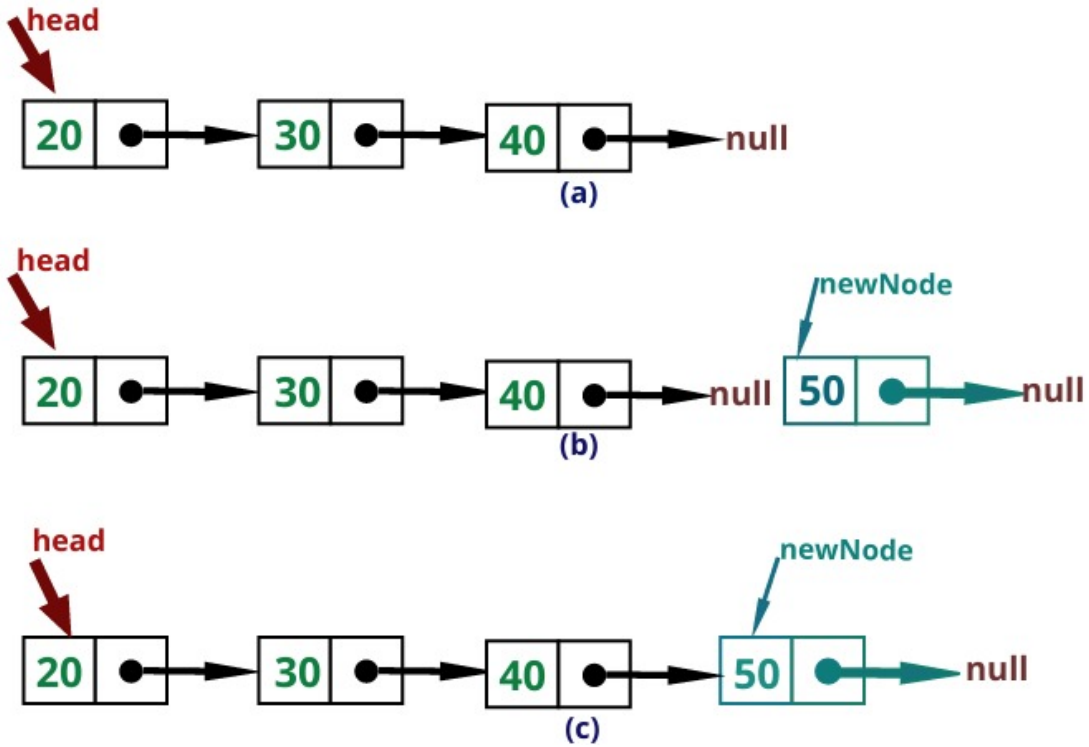
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Array
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications - Infix to Postfix, Reversal, Parenthesis Matching
  - Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertion at Front of SLL, Display the SLL

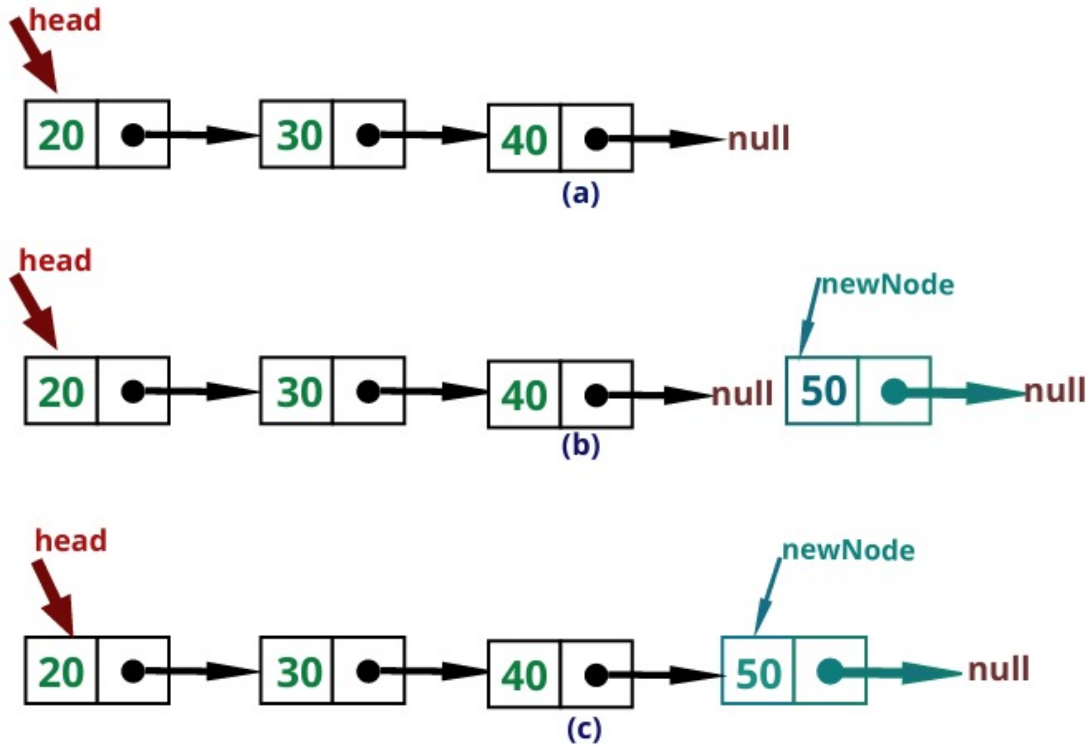
Today We Will See...

- Insertion at End of SLL

Insert a Node at End- Idea

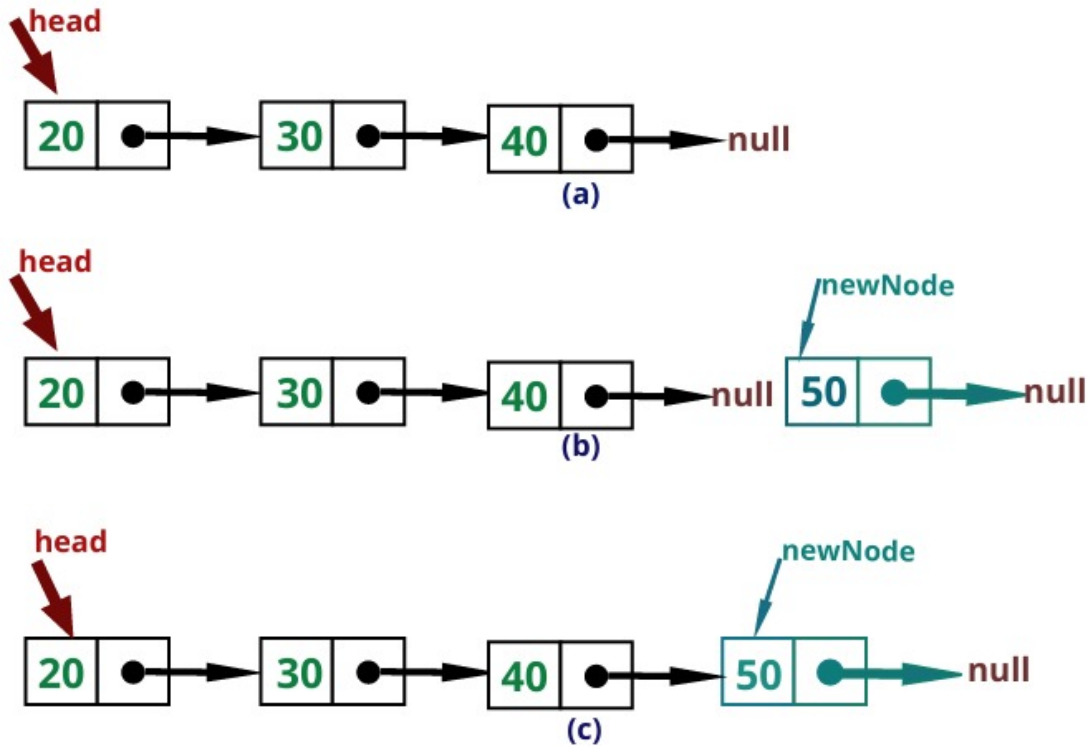


Insert a Node at End- Idea



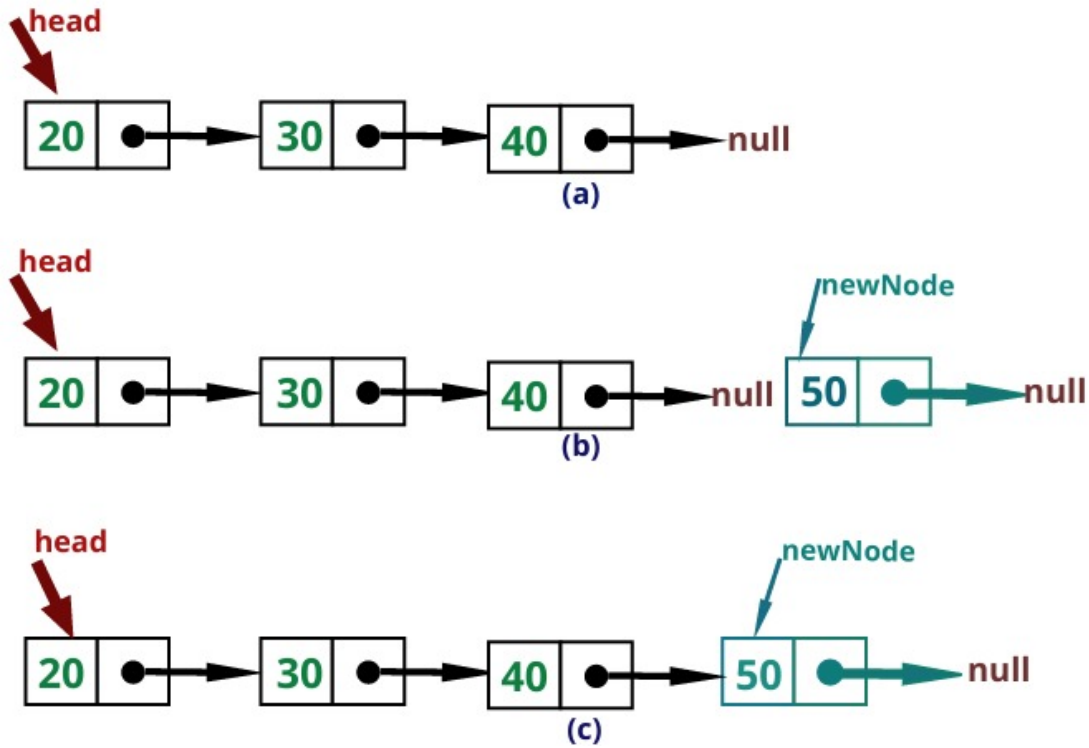
- Create a new node and put this value in the data part of the node

Insert a Node at End- Idea



- Create a new node and put this value in the data part of the node
- Make the node's next as NULL

Insert a Node at End- Idea



- Create a new node and put this value in the data part of the node
- Make the node's next as NULL
- Change the next pointer in the last node point to this new node

Insert a Node at End

```
\\let val contains the value of the new node to be inserted
struct node *newNode,*p;
newNode = (struct node *) malloc(sizeof(struct node));
newNode->data = val;
newNode->next = NULL;
if(head==NULL){
  head = newNode;
}else{
  p = head;
  while(p->next!=NULL){
    p=p->next;
  }
  p->next = newNode;
}
```

Lecture 22

We Saw & Will See

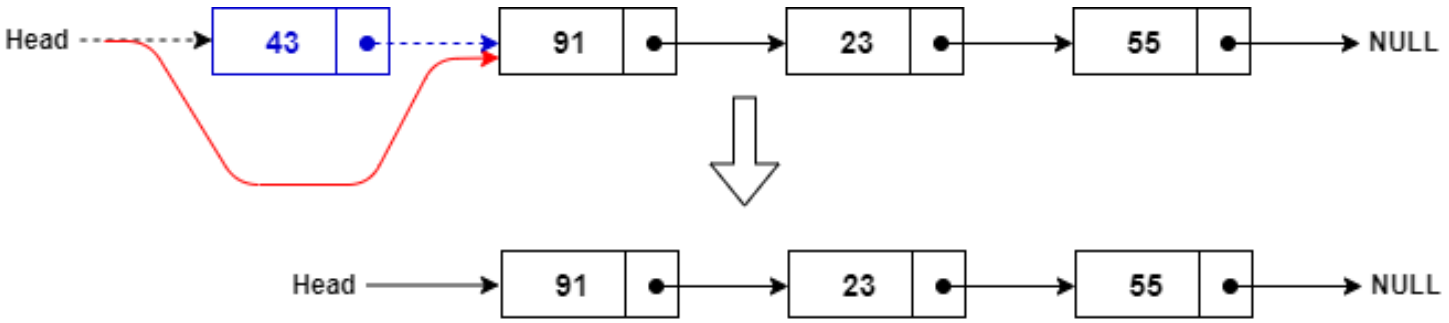
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Array
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications - Infix to Postfix, Reversal, Parenthesis Matching
  - Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertion at Front and End of SLL, Display the SLL

Today We Will See...

- Deletion at Front and End of SLL

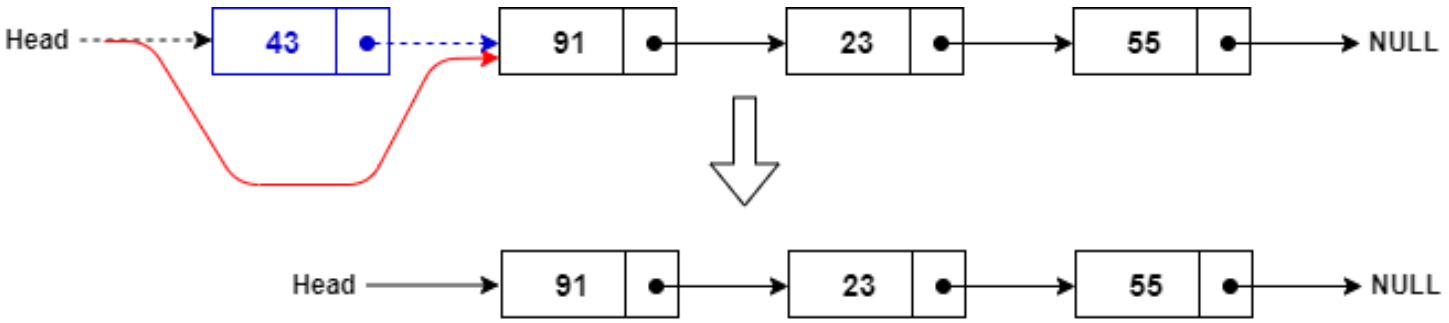
Delete a node at Front - Idea



Delete the First Node of a Linked List

- Change the head to the second node!

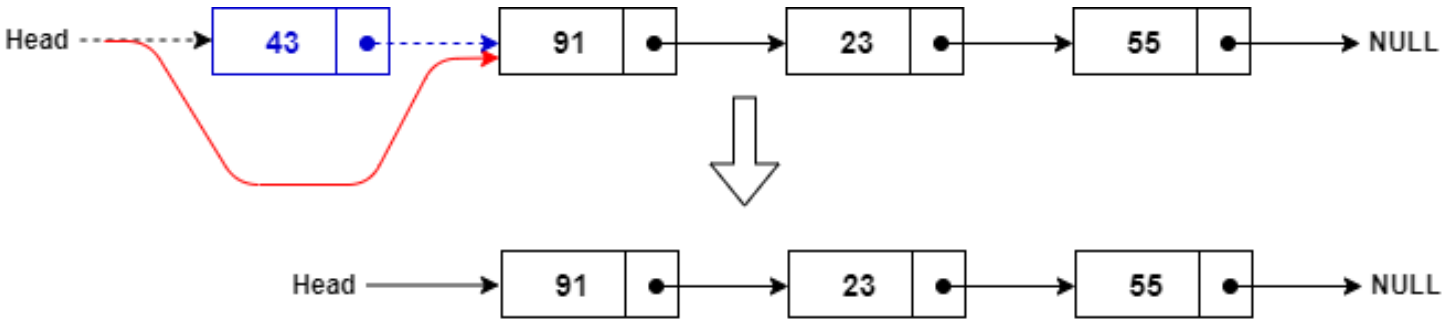
Delete a node at Front - Idea



Delete the First Node of a Linked List

- Change the head to the second node!
- What if head is NULL ?

Delete a node at Front - Idea



Delete the First Node of a Linked List

- Change the head to the second node!
- What if head is NULL ?
- What if there is only one node ?

Delete a node at Front

```
if(head!=NULL){
    head = head->next;
}
```

- Change the head to the second node! ✓

Delete a node at Front

```
if(head!=NULL){
    head = head->next;
}
```

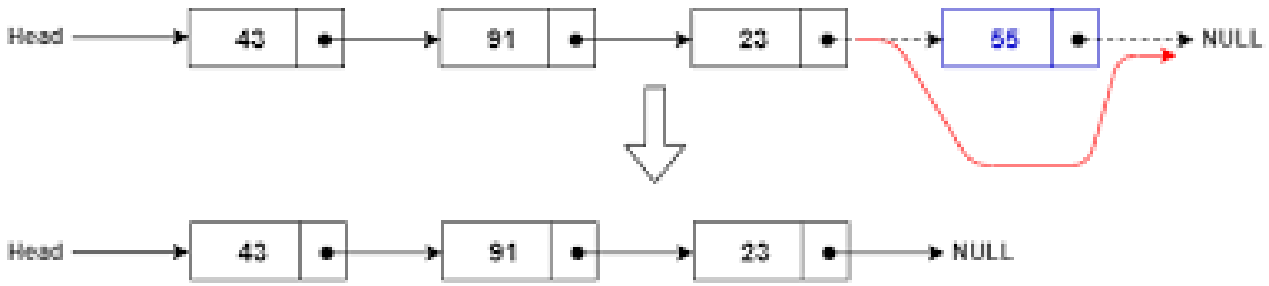
- Change the head to the second node! ✓
- What if head is NULL ? ✓

Delete a node at Front

```
if(head!=NULL){
    head = head->next;
}
```

- Change the head to the second node! ✓
- What if head is NULL ? ✓
- What if there is only one node ? ✓
  - head->next is NULL
  - So, head will become NULL after this operation!

Delete a node at End - Idea

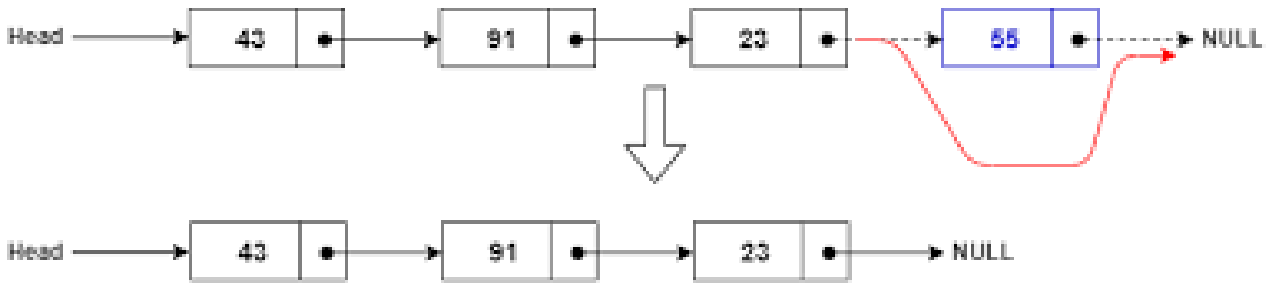


Delete the Last Node of a Linked List

qnaplus.com

- Find the second last node

Delete a node at End - Idea

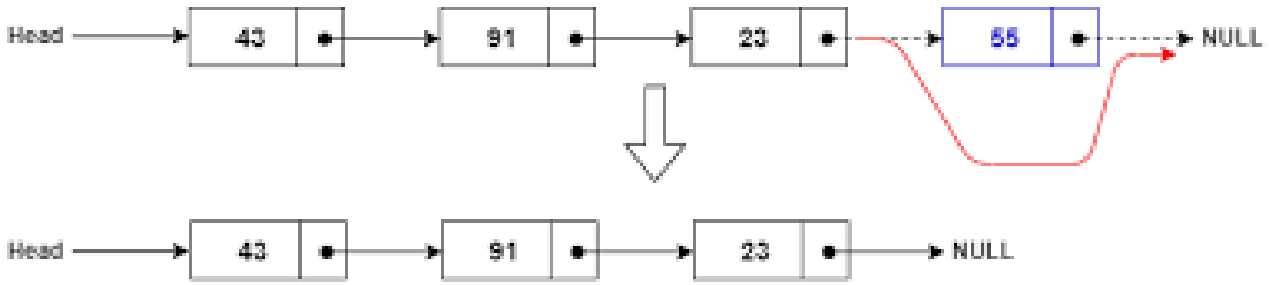


Delete the Last Node of a Linked List

qnaplus.com

- Find the second last nodeMake its next to NULL
- What if head is NULL ?

Delete a node at End - Idea



Delete the Last Node of a Linked List

qnaplus.com

- Find the second last nodeMake its next to NULL
- What if head is NULL ?
- What if there is only one node ?

Delete a node at End

```
struct node *p ;
 if(head!=NULL){
  if(head->next==NULL){
     head = NULL;
  }else{
     p=head;
     while(p->next->next!=NULL){
        p= p->next;
     }
     p->next =NULL;
  }
```

- What if head is NULL ? ✓

Delete a node at End

```
struct node *p ;
 if(head!=NULL){
  if(head->next==NULL){
     head = NULL;
  }else{
     p=head;
     while(p->next->next!=NULL){
        p= p->next;
     }
     p->next =NULL;
  }
```

- What if head is NULL ? ✓
- What if there is only one node ? Then head->next==NULL ✓

Delete a node at End

```
struct node *p ;
 if(head!=NULL){
  if(head->next==NULL){
     head = NULL;
  }else{
     p=head;
     while(p->next->next!=NULL){
        p= p->next;
     }
     p->next =NULL;
  }
```

- What if head is NULL ? ✓
- What if there is only one node ? Then head->next==NULL ✓
- Find the second last node. Make its next to NULL ✓

Delete a node at End

```
struct node *p ;
 if(head!=NULL){
  if(head->next==NULL){
     head = NULL;
  }else{
     p=head;
     while(p->next->next!=NULL){
        p= p->next;
     }
     p->next =NULL;
  }
```

- What if head is NULL ? ✓
- What if there is only one node ? Then head->next==NULL ✓
- Find the second last node. Make its next to NULL ✓

Lecture 23

We Saw & Will See

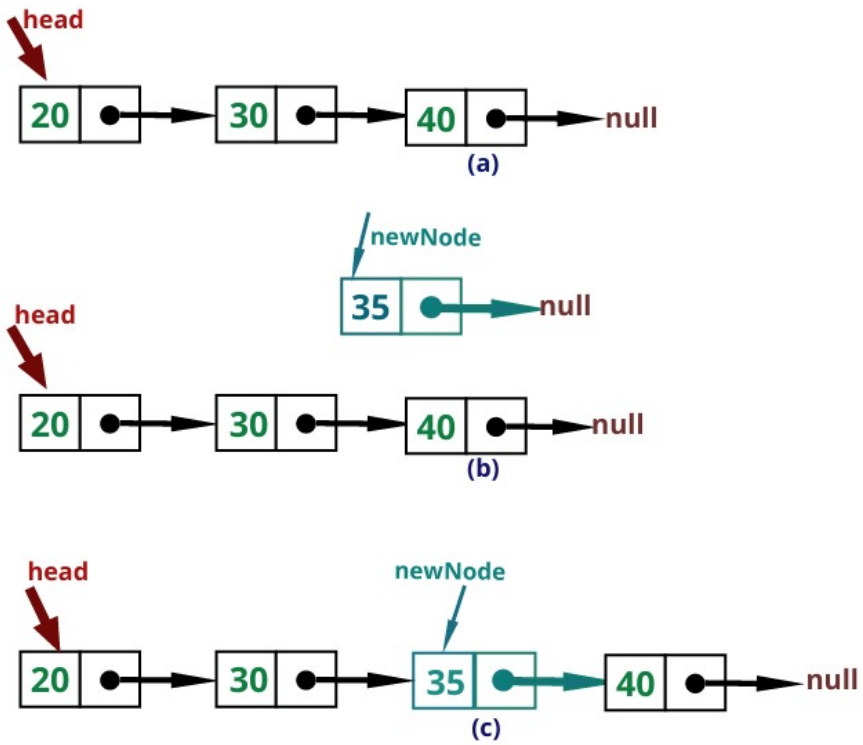
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Array
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications - Infix to Postfix, Reversal, Parenthesis Matching
  - Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertion at Front and End of SLL, Display the SLL
  - Deletion at Front and End of SLL

Today We Will See...

- Insert After a specified Node in SLL

Insert After a Node- Idea



- Reach the node after which we need to insert the new node.
- Set the next of the new node equal to the next of this reached node.
- Set the next of reached node as the new node.

Insert After Node whose data==key

```
struct node *p,*newNode;
newNode = (struct node *) malloc( sizeof(struct node));
newNode->data = val;
p = head;
while(p->data!=key){
   p= p->next;
}
newNode->next = p->next;
p->next = newNode;
```

- the above code assumes that there exists a node whose data part contains key

Lecture 24

We Saw & Will See

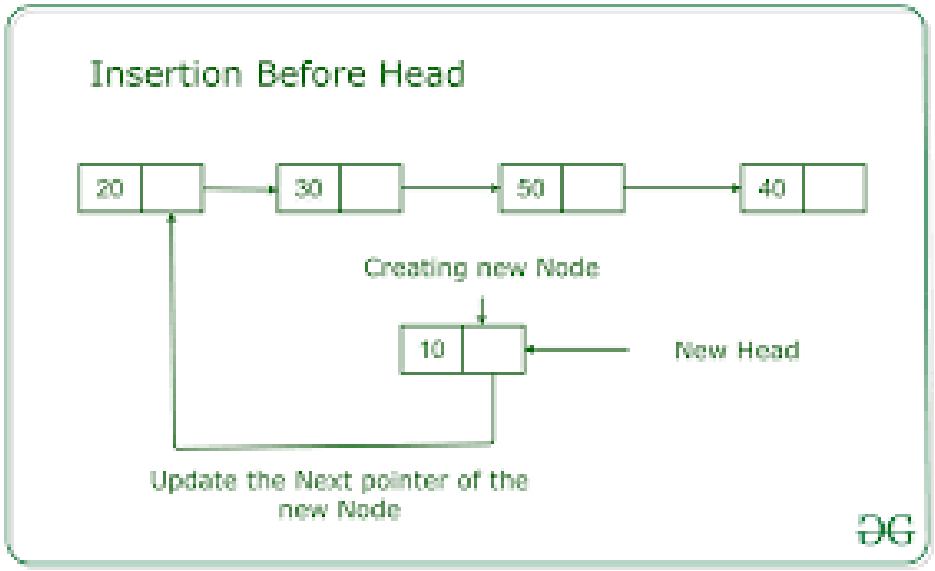
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Array
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications, Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertion,Deletion at Front and End of SLL, Display the SLL
  - Insert After a specified Node in SLL

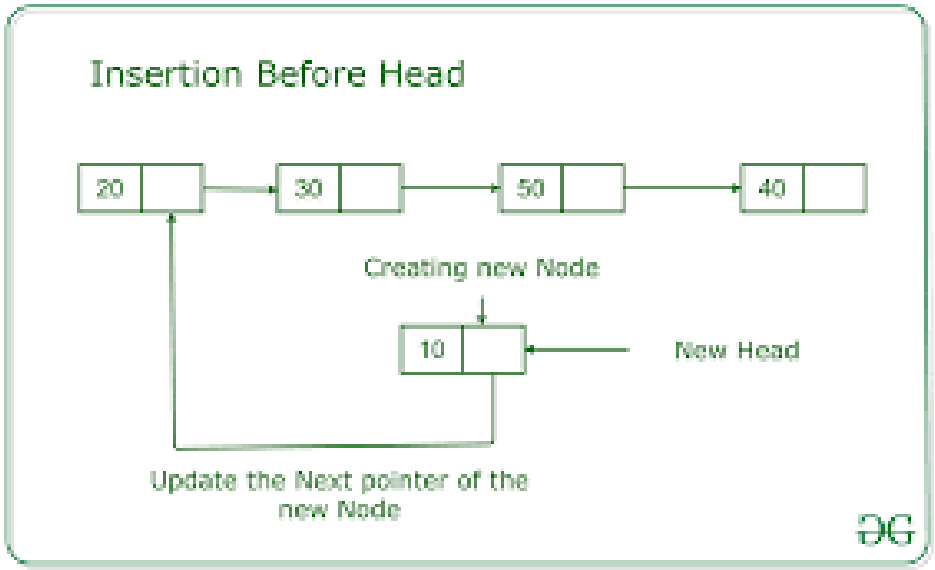
Today We Will See...

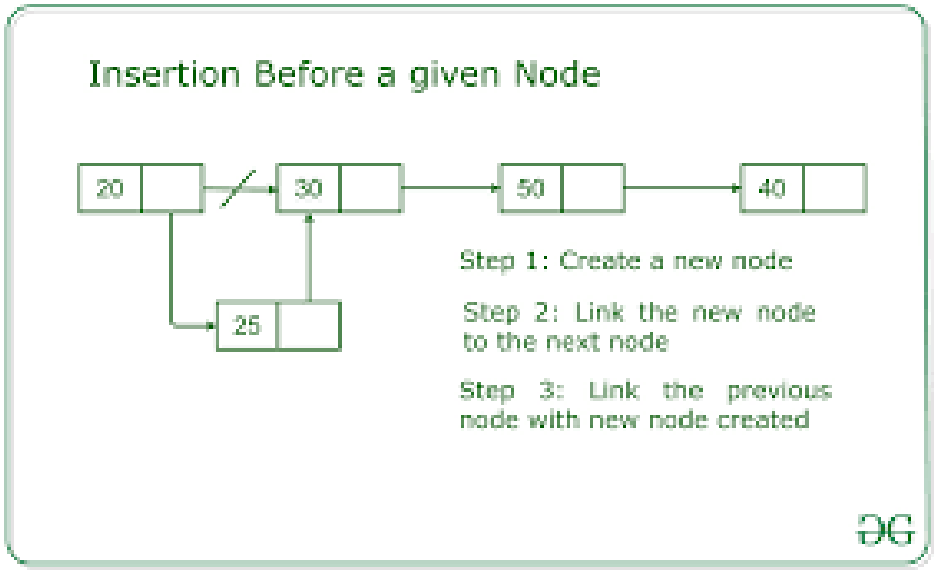
- Insert Before a specified Node in SLL
- Delete the specified Node in SLL
- Stack and Queue using Linked List

Insert Before a Node- Idea

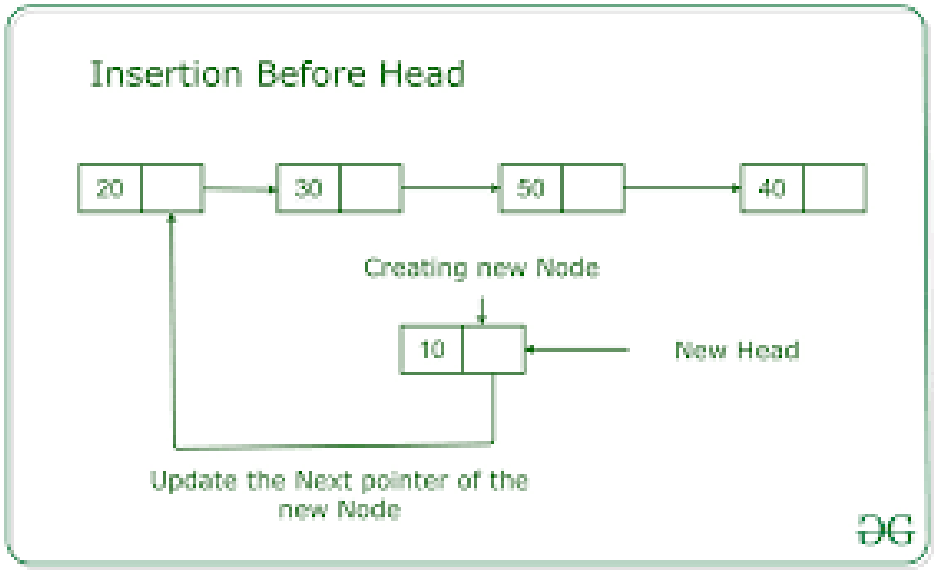


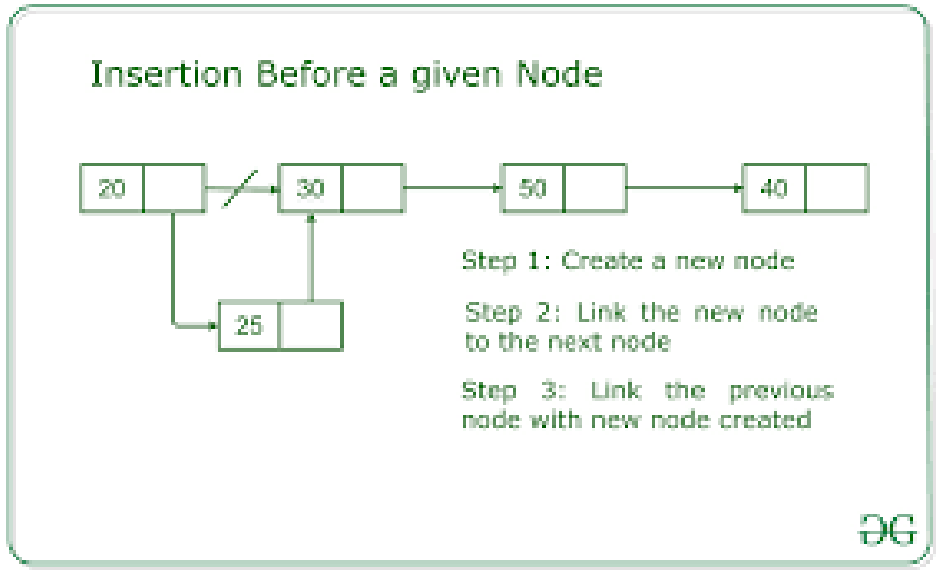
Insert Before a Node- Idea





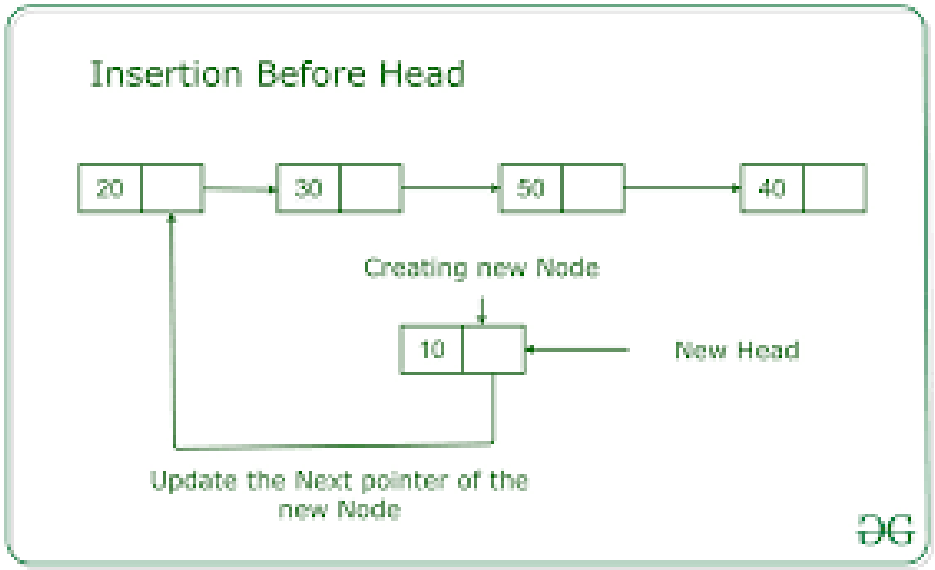
Insert Before a Node- Idea

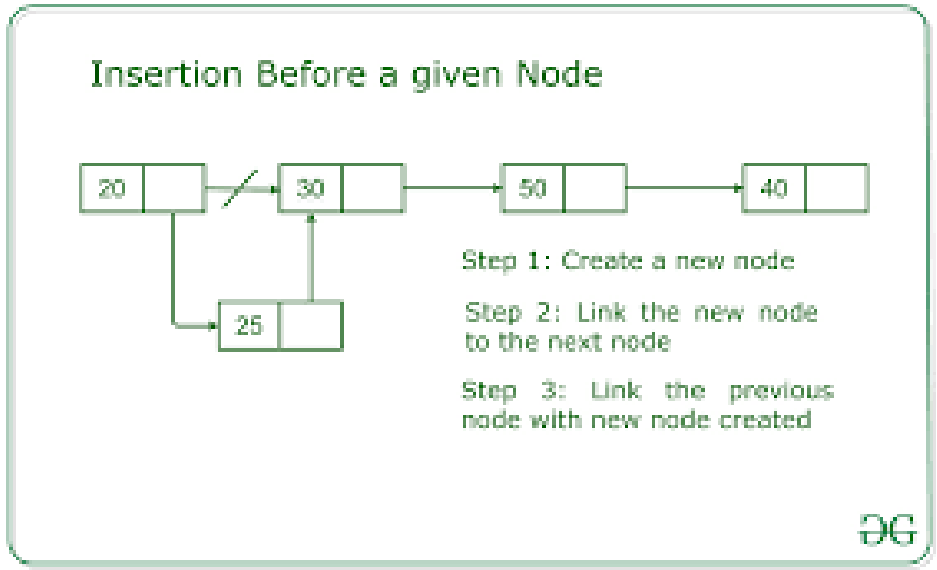




- Reach one node prior to the node before which we need to insert the new node.

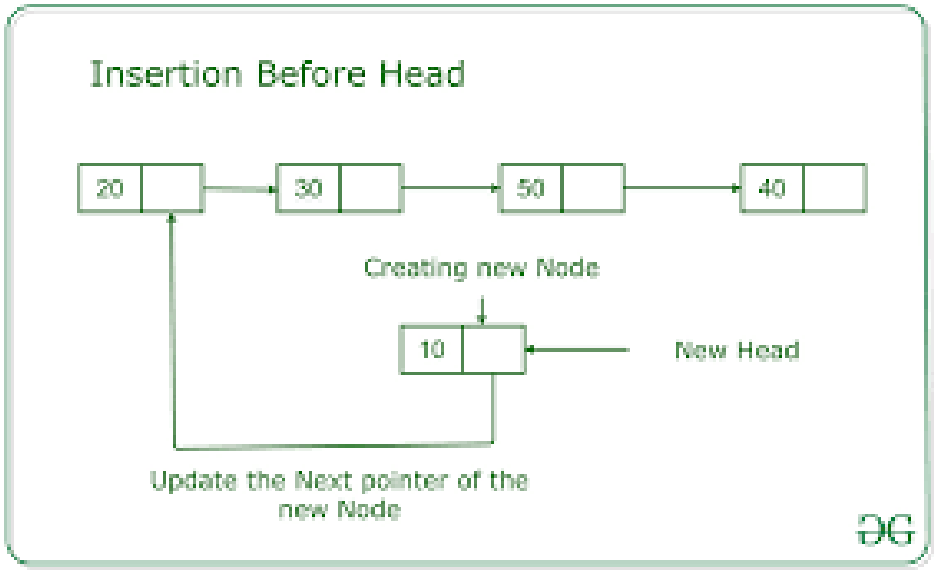
Insert Before a Node- Idea

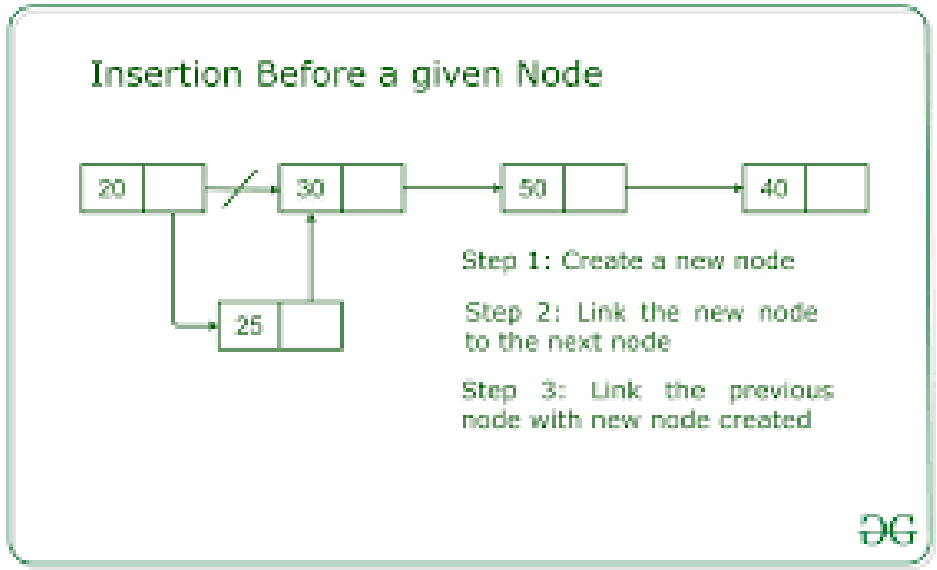




- Reach one node prior to the node before which we need to insert the new node.
- Do the steps of Insert After this reached node!

Insert Before a Node- Idea





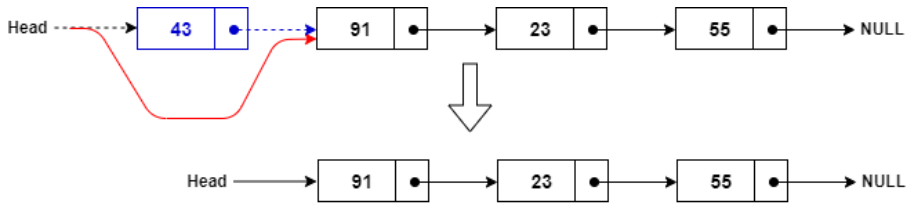
- Reach one node prior to the node before which we need to insert the new node.
- Do the steps of Insert After this reached node!
- Special Case : Insert before head!

Insert Before Node whose data==key

```
struct node *p,*newNode;
newNode = (struct node *) malloc( sizeof(struct node));
newNode->data = val;
if(head->data==key){
  newNode->next = head;
  head = newNode;
else{
  p = head;
  while(p->next->data!=key){
    p= p->next;
  }
  newNode->next = p->next;
  p->next = newNode;
}
```

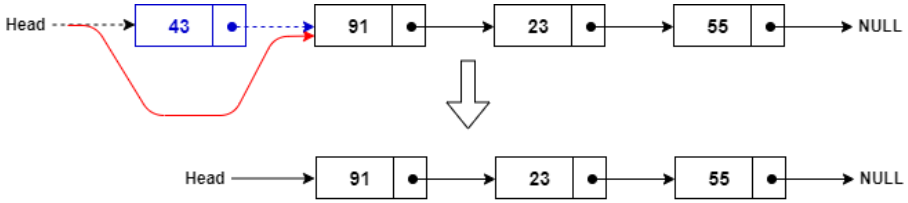
- code assumes that there exists a node whose data part contains key

Delete a Node whose data == key : Algorithm Idea

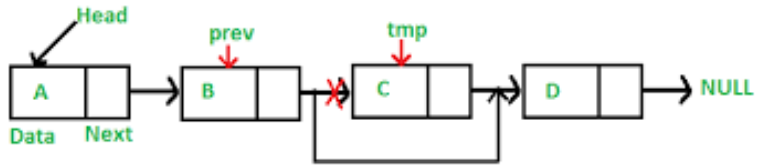


Delete the First Node of a Linked List

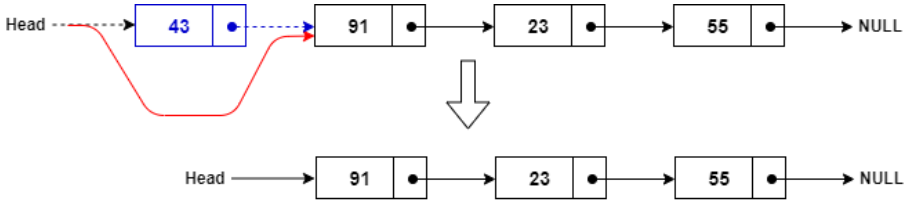
Delete a Node whose data == key : Algorithm Idea



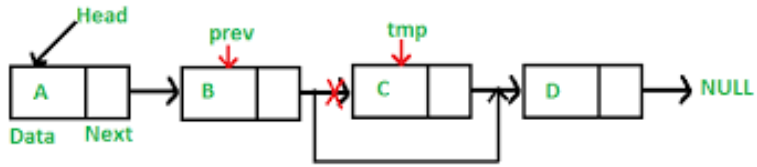
Delete the First Node of a Linked List



Delete a Node whose data == key : Algorithm Idea

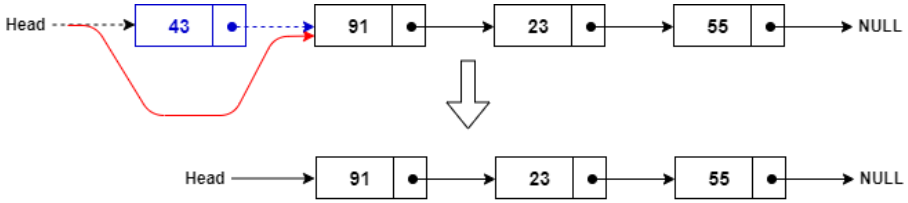


Delete the First Node of a Linked List



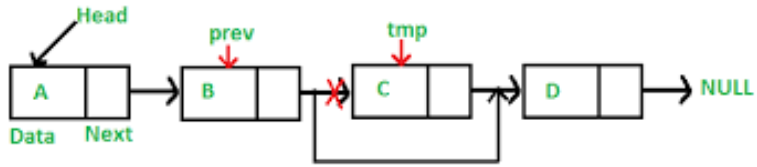
- Reach one node prior to the node whose data part equals key.

Delete a Node whose data == key : Algorithm Idea



Delete the First Node of a Linked List

qnaplus.com



- Reach one node prior to the node whose data part equals key.
- change the next to point to next of next node!
- Spl. case : first node's data is key

Delete Node whose data==key

```
if(head->data==key){
  head = head->next;
else{
  p = head;
  while(p->next->data!=key){
    p= p->next;
  }
  p->next = p->next->next;
}
```

- code assumes that there exists a node whose data part contains key

Stack and Queue using Linked List in 1 minute!

Stack

- Push(int x) => insertAtFront(int x)

Stack and Queue using Linked List in 1 minute!

Stack

- Push(int x) => insertAtFront(int x)
- Pop() => deleteFromFront()

Stack and Queue using Linked List in 1 minute!

Stack

- Push(int x) => insertAtFront(int x)
- Pop() => deleteFromFront()

Queue

- Enqueue(int x) => insertAtEnd(int x)

Stack and Queue using Linked List in 1 minute!

Stack

- Push(int x) => insertAtFront(int x)
- Pop() => deleteFromFront()

Queue

- Enqueue(int x) => insertAtEnd(int x)
- DeQueue() => deleteFromFront()

Stack and Queue using Linked List in 1 minute!

Stack

- Push(int x) => insertAtFront(int x)
- Pop() => deleteFromFront()

Queue

- Enqueue(int x) => insertAtEnd(int x)
- DeQueue() => deleteFromFront()

Lecture 25

We Saw & Will See

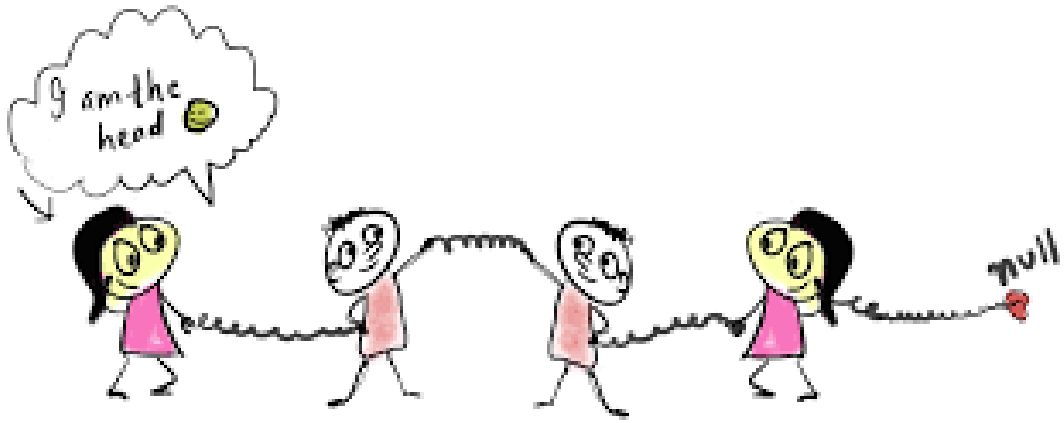
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Arrays & Linked Lists
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications, Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertion,Deletion at Front and End of SLL, Display the SLL
  - Insert After/Before a specified Node in SLL
  - Delete the specified Node in SLL
  - Stack and Queue using Linked List

Today We Will See...

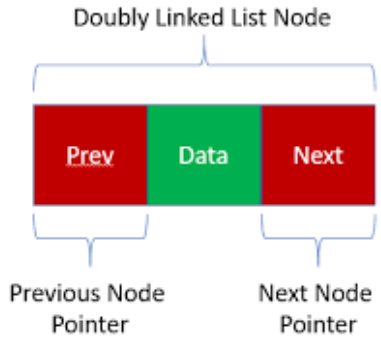
- Doubly Linked List

Doubly Linked List - Pictorial Representation



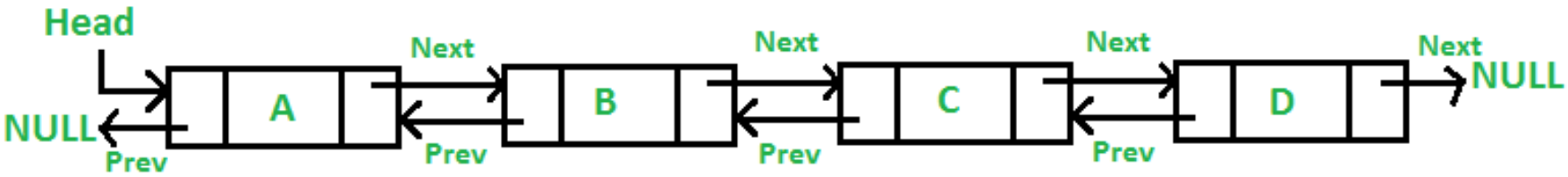
- Everyone knows who is before and who is after!

Doubly Linked List Node



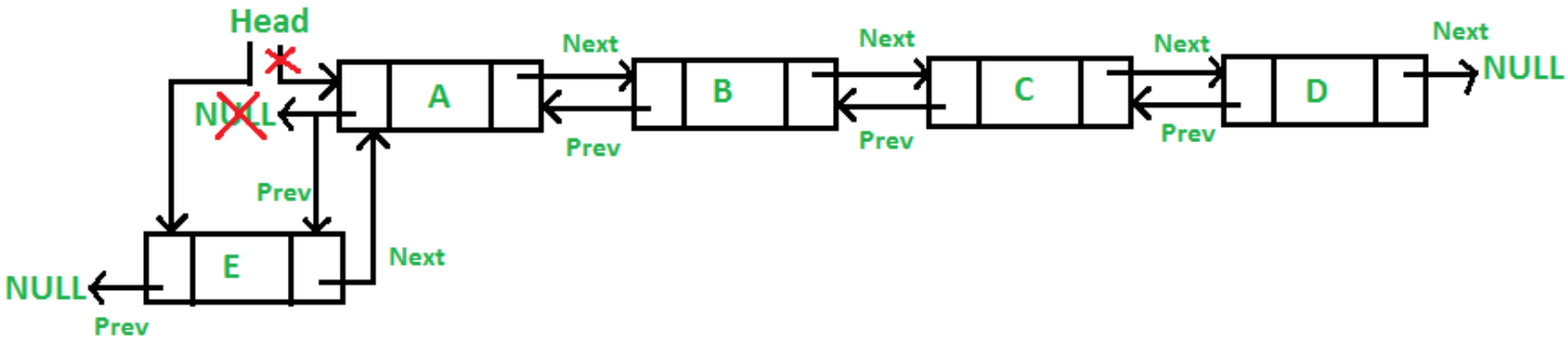

```
struct dnode{
    struct dnode *prev;
    int data;
    struct dnode *next;
}*head=NULL;
```

Doubly Linked List(DLL)



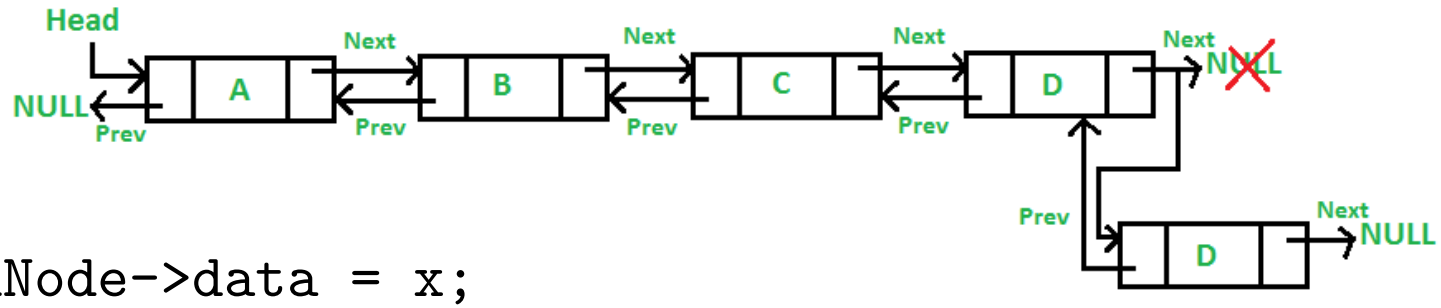
- head contains the address of the first node in the doubly linked list
- every node contains a previous, data and next parts
- previous, next part contains the address of the previous and next nodes respectively
- prev is NULL for the first node and next is NULL for the last node

Insert_At_Front_DLL(int x)



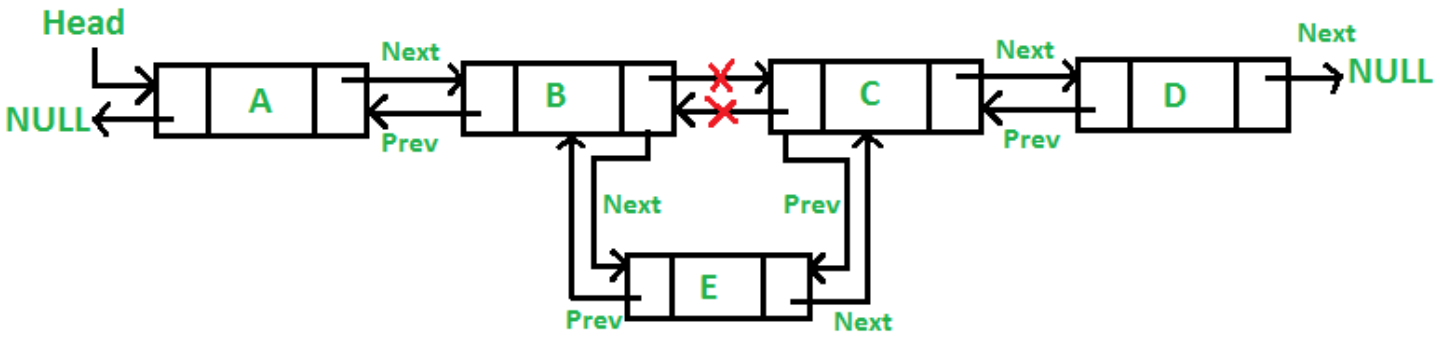

```
newdNode = (struct dnode *)(malloc sizeof(struct dnode));
newdNode->data = x;
newdNode->prev = NULL;
newdNode->next = head;
head = newdNode;
```

Insert_At_End_DLL(int x)




```
newdNode->data = x;
if(head==NULL){
   newdNode->prev =newNode->next = NULL;
   head = newNode;
 }else{
   p=head;
   while(p->next!=NULL)
       p= p->next;
   p->next = newdNode;
   newdNode->prev = p;
   newdNode->next = NULL;
}
```

Insert_After_DLL(int x, int key)

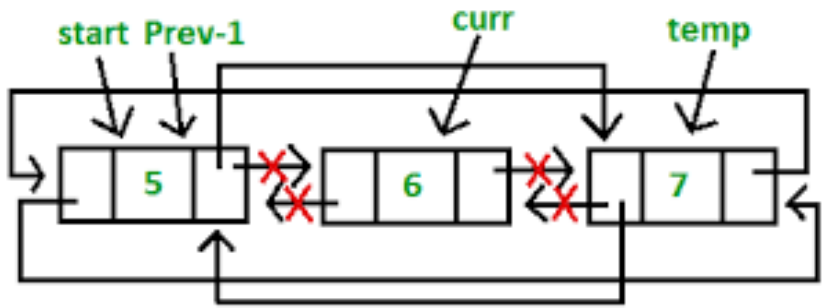


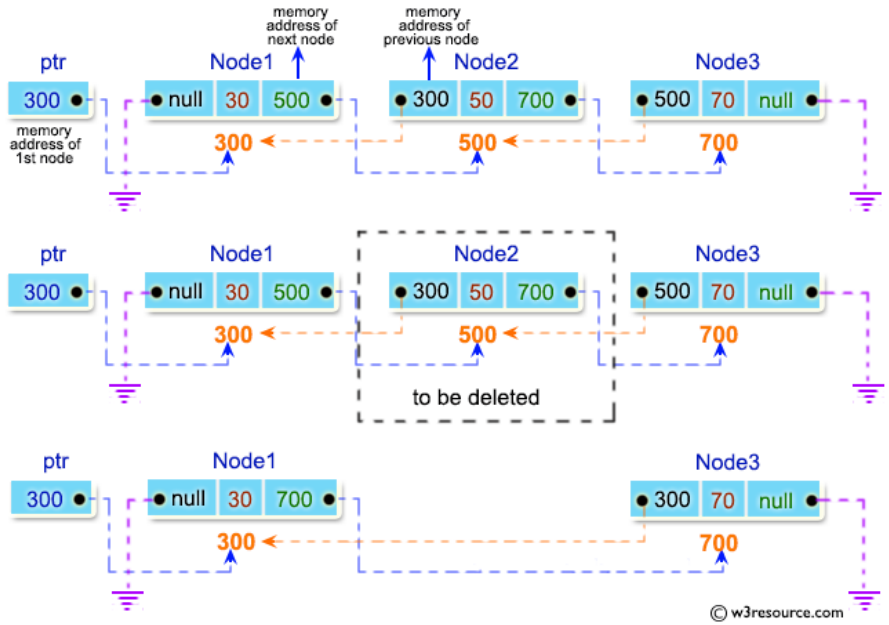

```
newdNode->data = x;
p=head;
while(p->data!=key)
   p=p->next;
newdNode->prev = p;
newdNode->next = p->next;
newdNode->next->prev = newdNode;
p->next = newdNode;
```

Insert_Before_DLL(int x, int key)

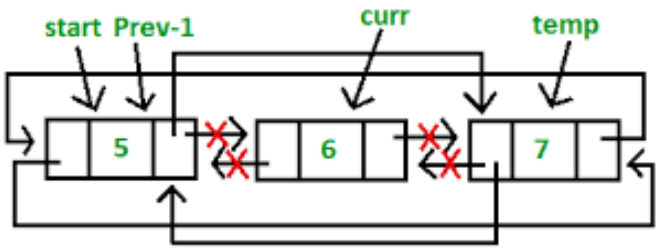
- can use the same method, since it is easy to reach one node before using the prev link. Then use insert_After_DLL(int x, int key)

Delete a Node in Doubly Linked List





Delete_DLL(int key)




```
if(head->data == key){
   head->next->prev = NULL;
   head = head->next;
 }else {
    p = head;
    while(p->data!=key)
       p=p->next;
    if(p->next!=NULL){
       p->next->prev = p->prev;
       p->prev->next = p->next;
    } else {
       p->prev->next = NULL;
    }
 }
```

* code assumes that there is at least two nodes in the linked list.

Lecture 26

We Saw & Will See

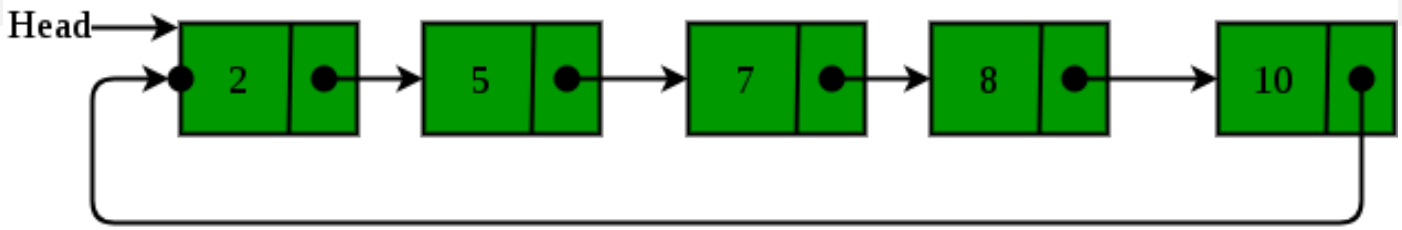
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Arrays & Linked Lists
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications, Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertion,Deletion at Front and End of SLL, Display the SLL
  - Insert After/Before a specified Node in SLL
  - Delete the specified Node in SLL
  - Stack and Queue using Linked List

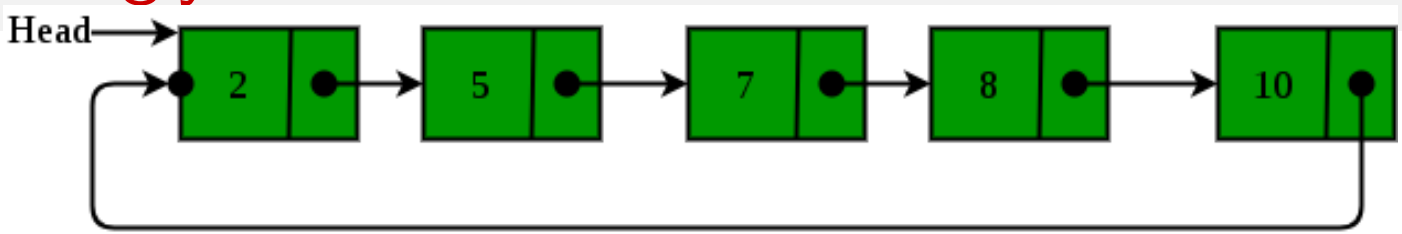
Today We Will See...

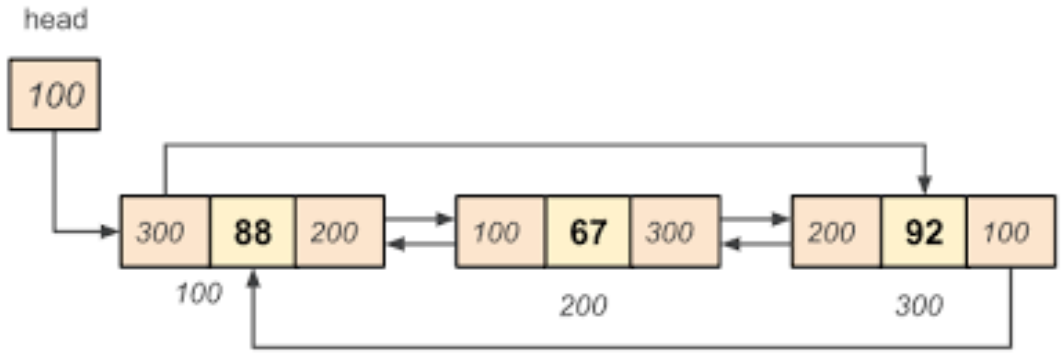
- Circular Linked Lists

Circular Singly Linked List

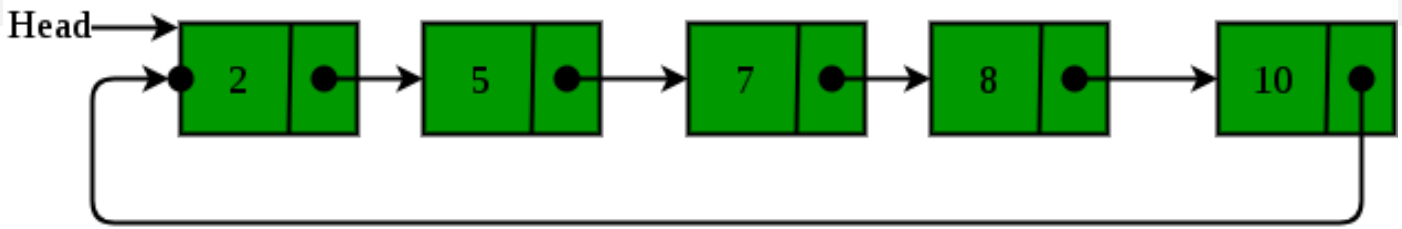


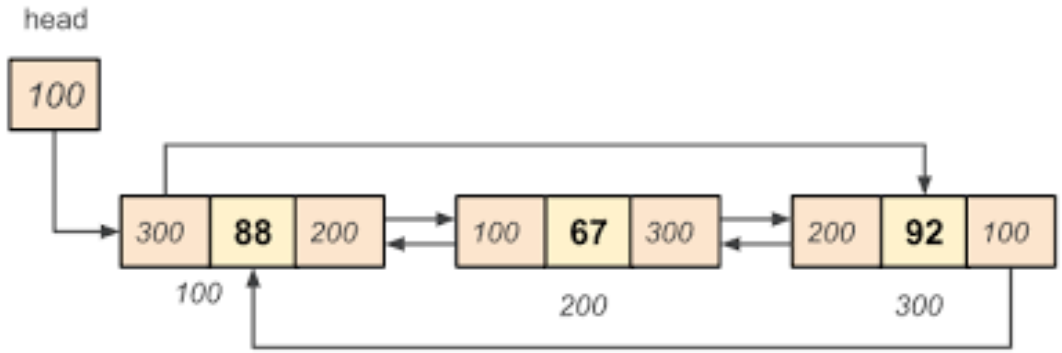
Circular Singly Linked List





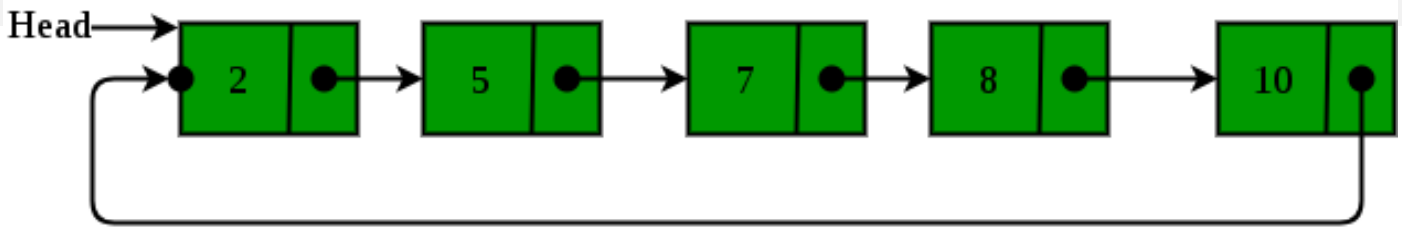
Circular Singly Linked List

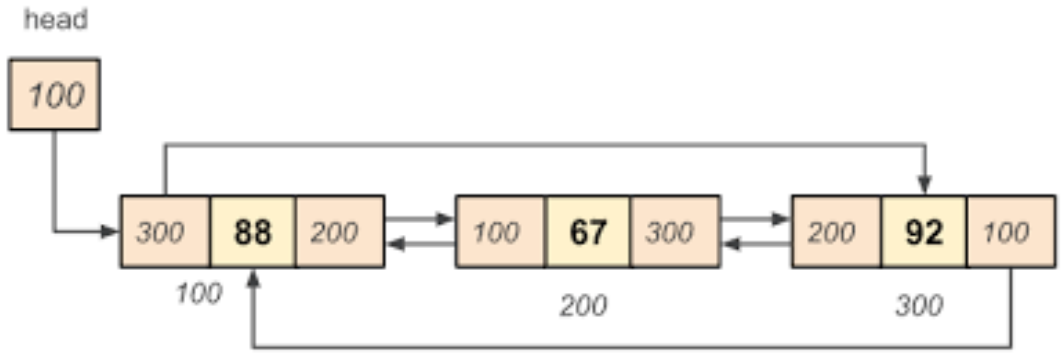




- very similar to usual singly linked list.

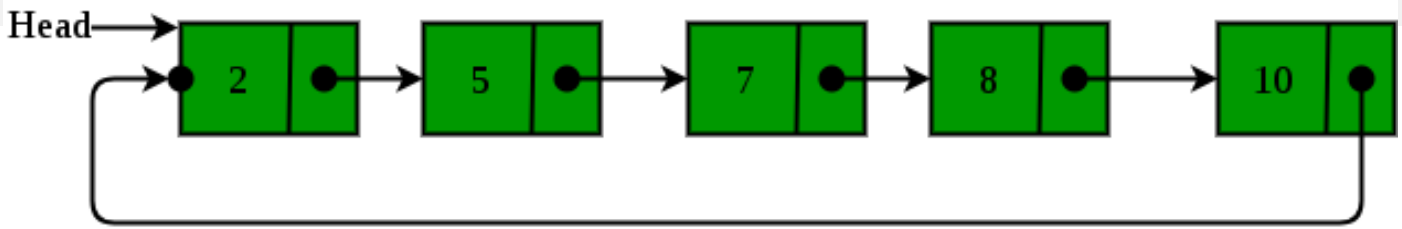
Circular Singly Linked List

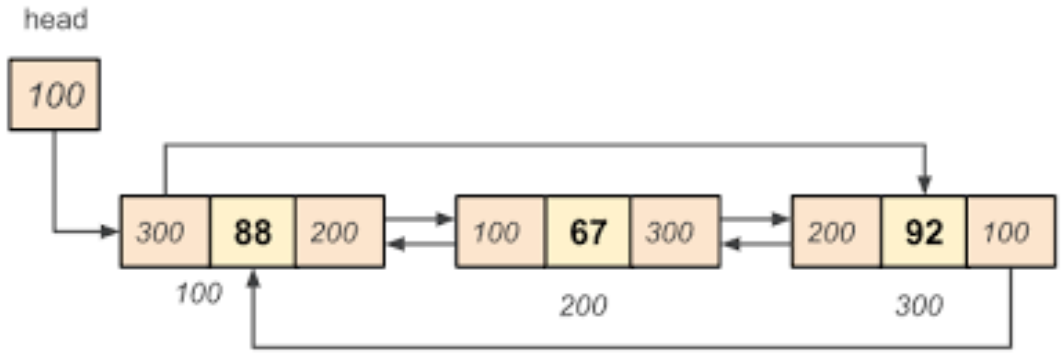




- very similar to usual singly linked list.
- no node's next is NULL.

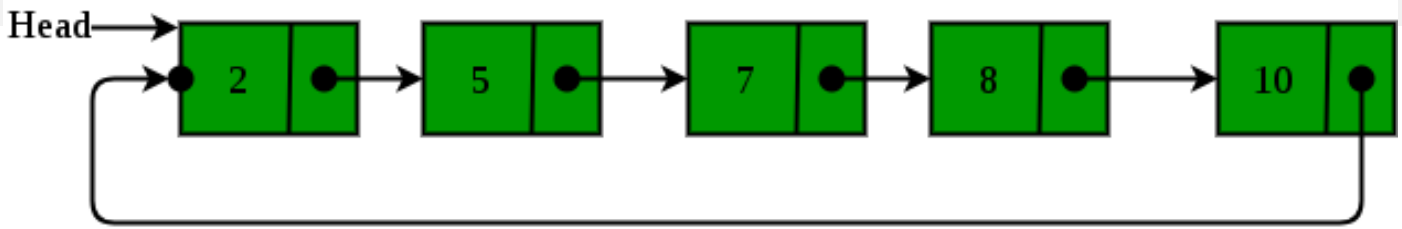
Circular Singly Linked List

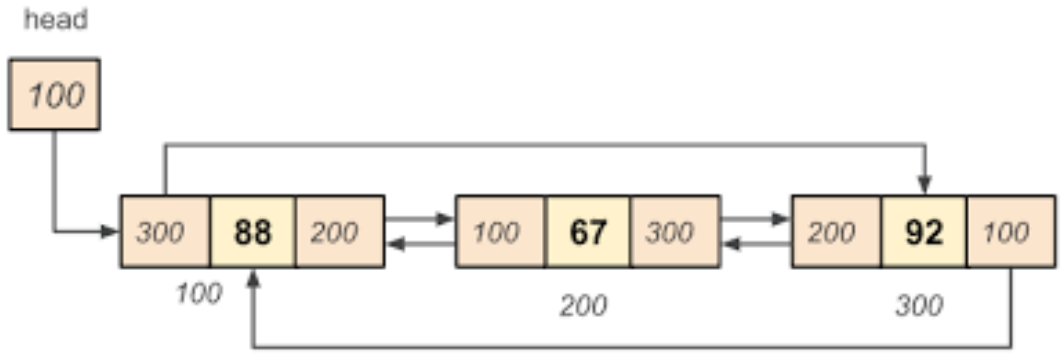




- very similar to usual singly linked list.
- no node's next is NULL.
- last node's next points to first node (head).

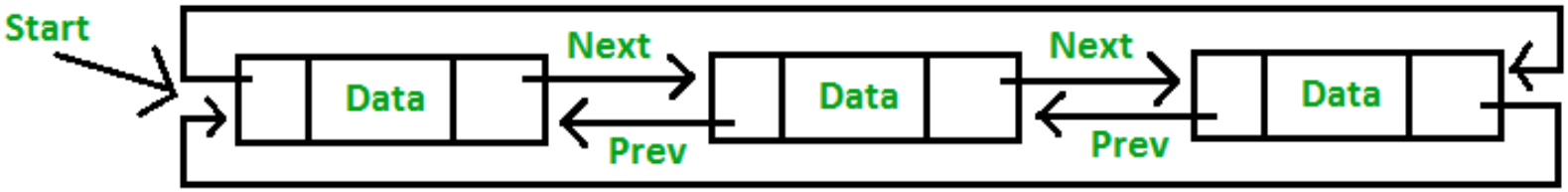
Circular Singly Linked List



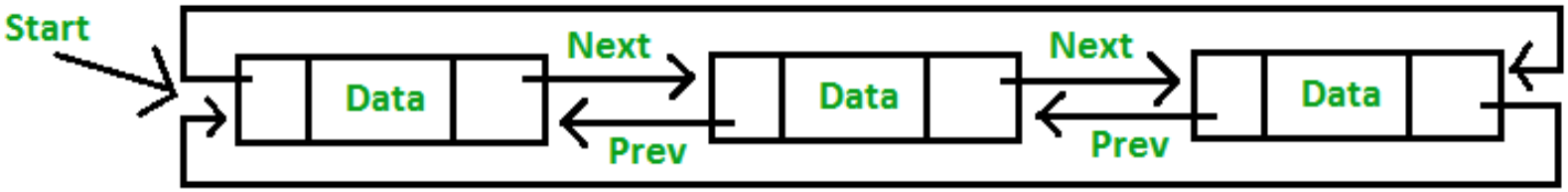


- very similar to usual singly linked list.
- no node's next is NULL.
- last node's next points to first node (head).
- Insertion, Deletion operations similar to that of singly linked list.

Circular Doubly Linked List

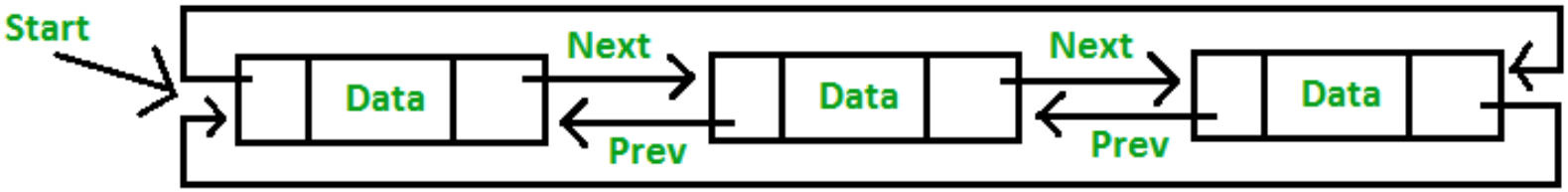


Circular Doubly Linked List



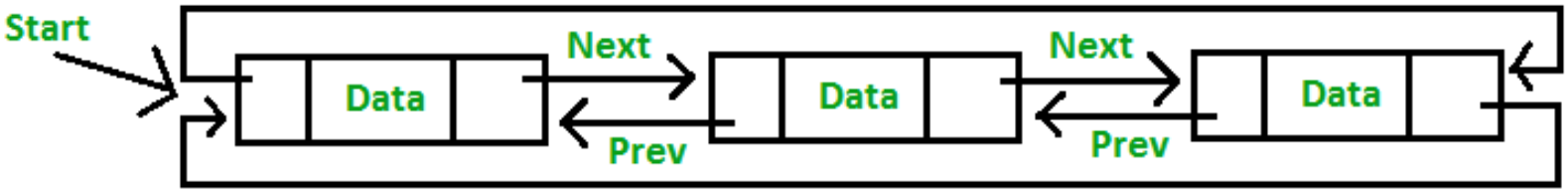
- very similar to usual doubly linked list.

Circular Doubly Linked List



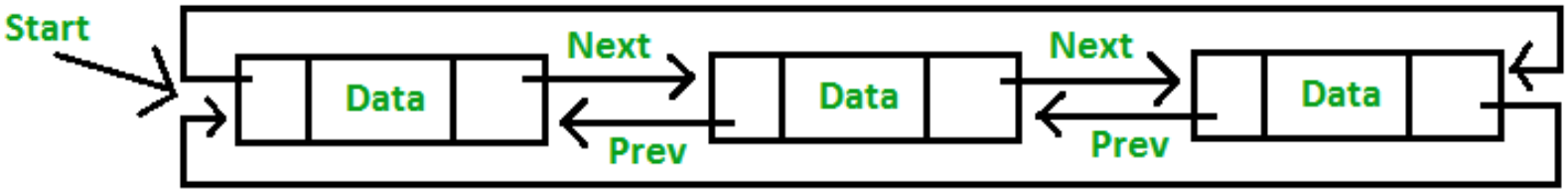
- very similar to usual doubly linked list.
- last node's next points to first node (head).

Circular Doubly Linked List



- very similar to usual doubly linked list.
- last node's next points to first node (head).
- first node's(head) prev points to last node.

Circular Doubly Linked List



- very similar to usual doubly linked list.
- last node's next points to first node (head).
- first node's(head) prev points to last node.
- Insertion, Deletion operations similar to that of doubly linked list.

Lecture 27

We Saw & Will See

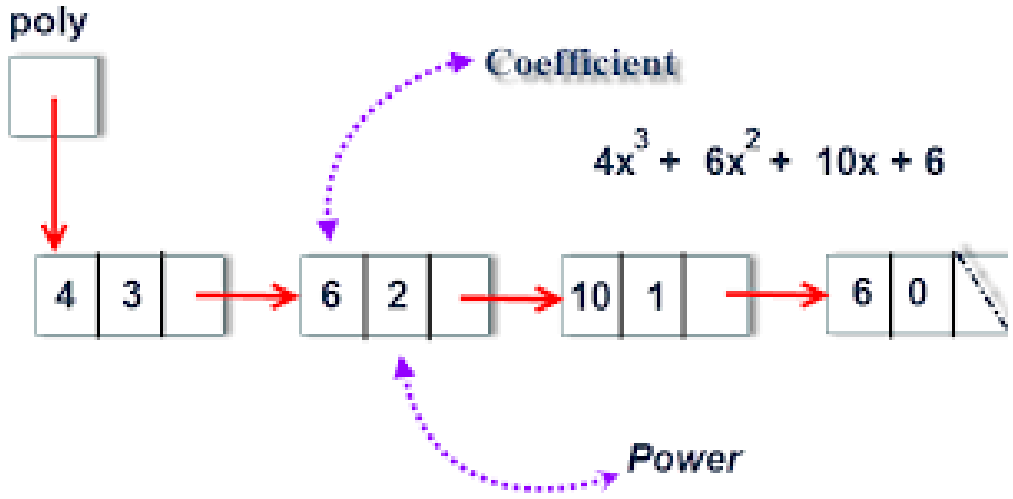
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack, Queue, Circular Queue using Arrays & Linked Lists
  - Infix, Prefix and Postfix, Evaluation of Postfix Expression
  - Stack Applications, Double Ended Queue, Priority Queue
- Module 2 : Linked Lists
  - Singly Linked List, Doubly Linked List, Circular Linked List
  - Insertions and Deletions in SLL, DLL and CLLs
  - Stack and Queue using Linked List

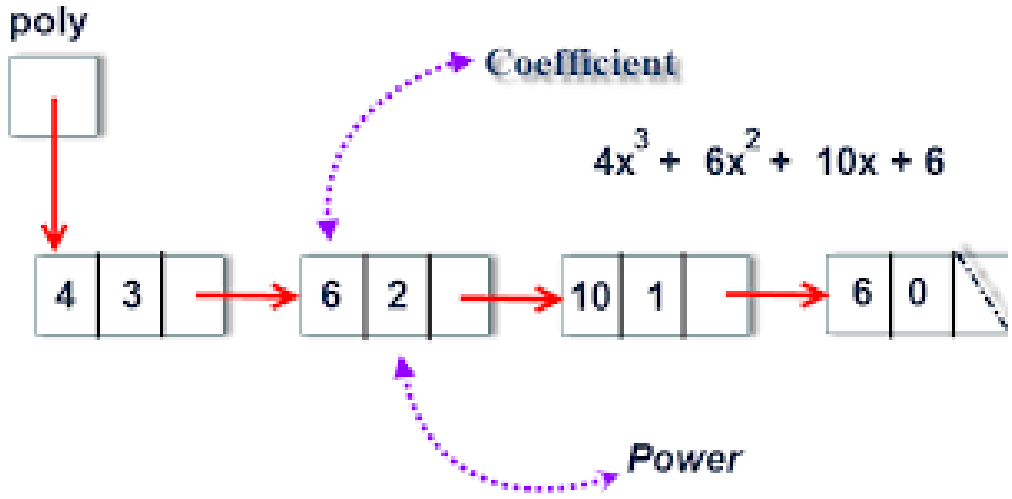
Today We Will See...

- Polynomials using Linked List
- Memory Allocation : First Fit, Best Fit, Worst Fit & Next Fit

Polynomials using Linked Lists

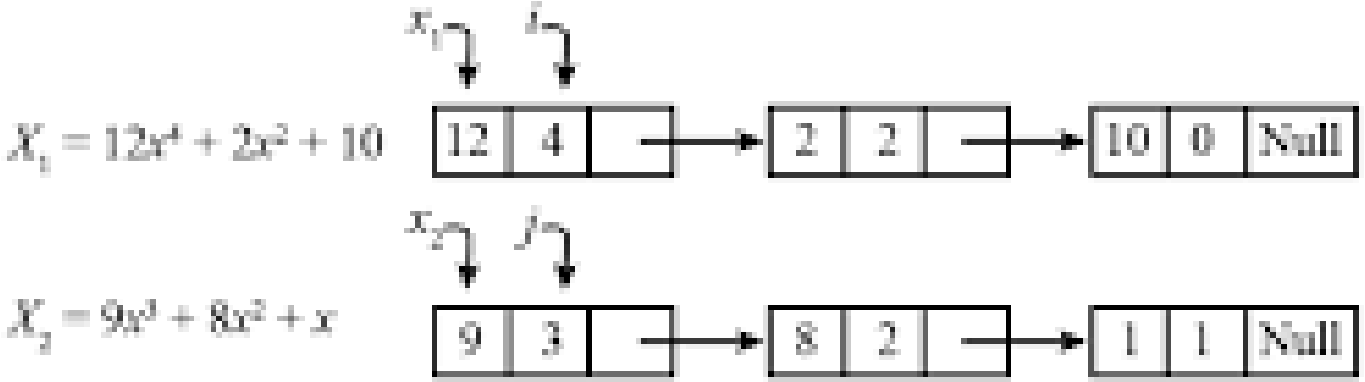


Polynomials using Linked Lists



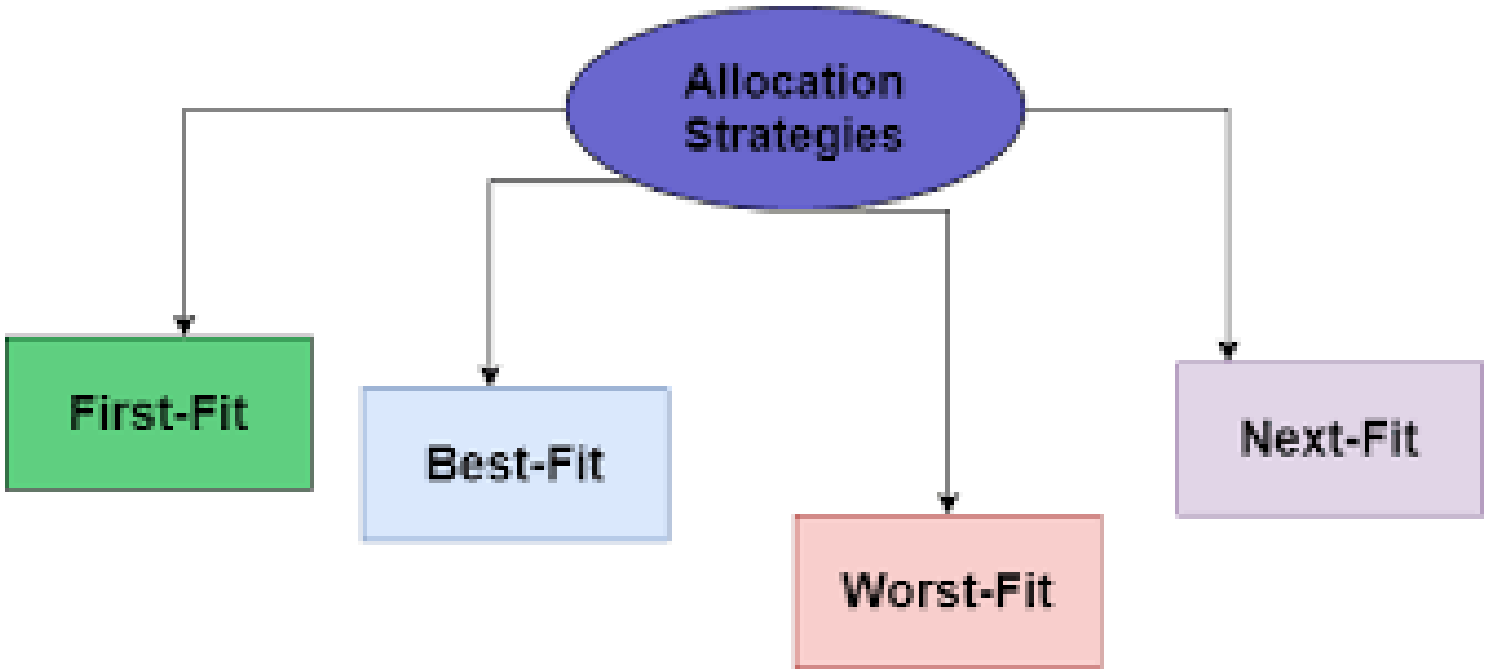

```
struct pnode{
    int coeff;
    int pow;
    struct node *next;
};
struct pnode *poly=NULL;
```

Polynomial Addition

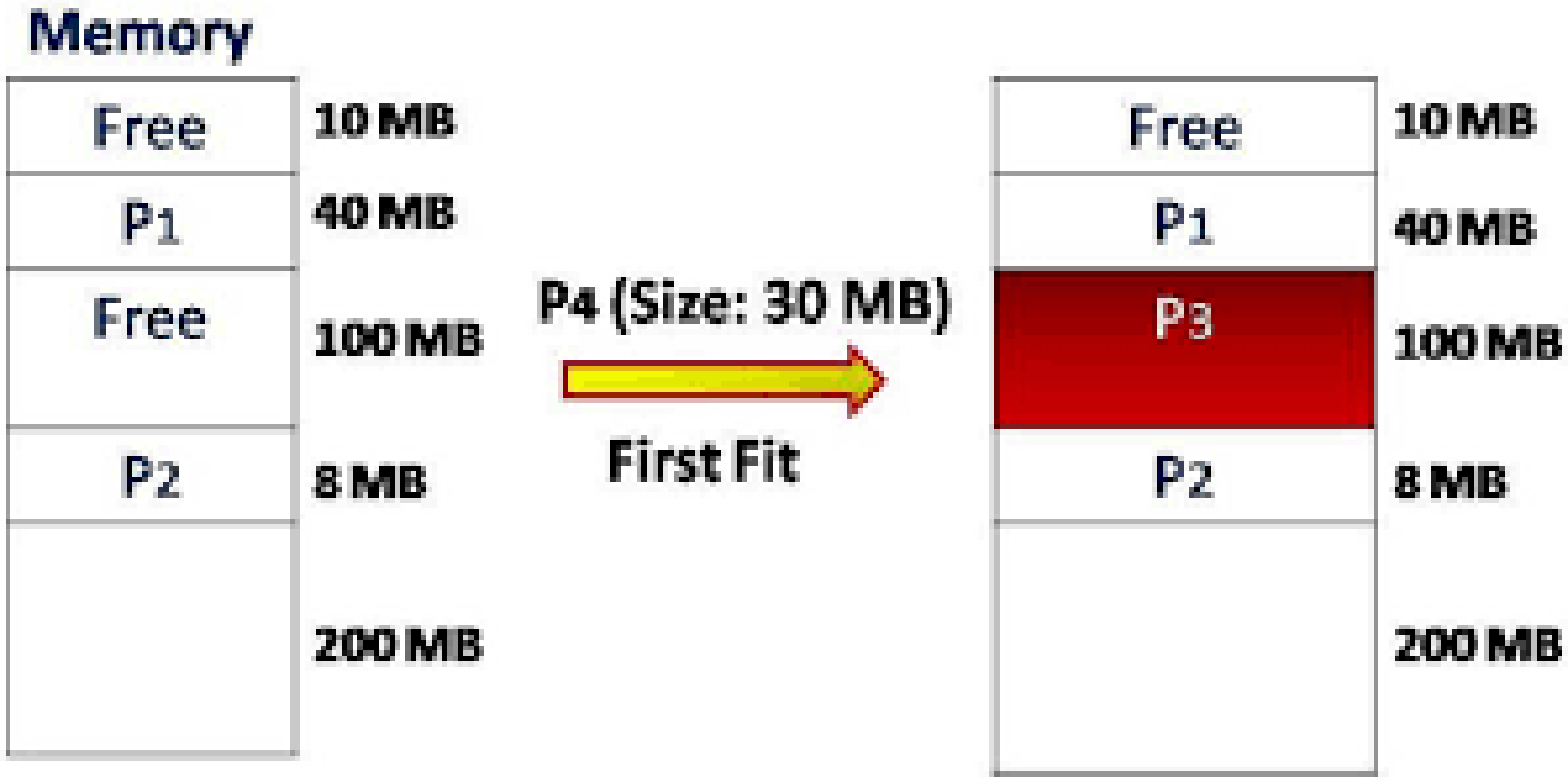


The resultant liked list :-

Memory Allocation



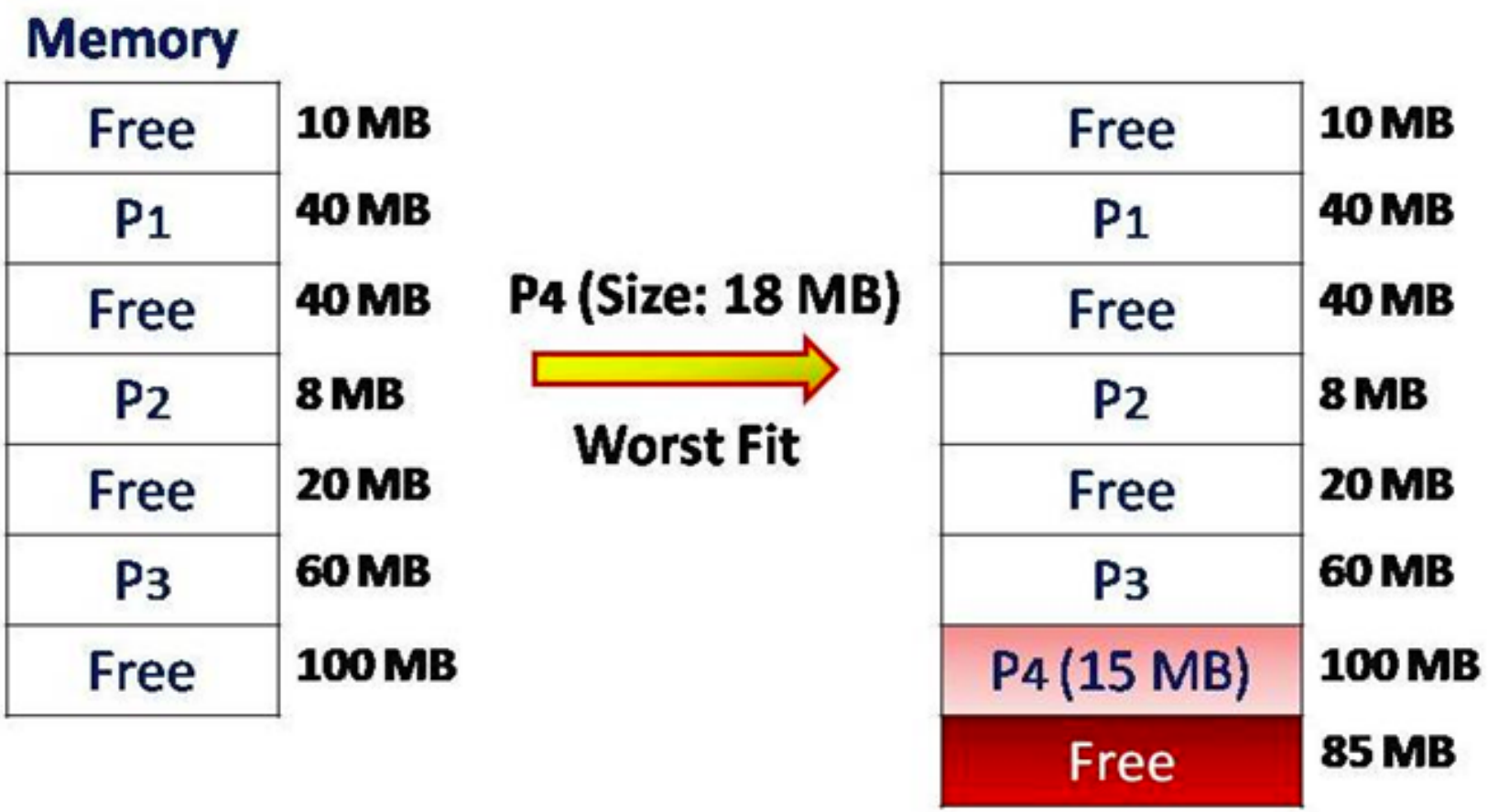
First Fit


Memory
Free
10 MB
P1
40 MB
Free
100 MB
P2
8 MB
200 MB
P4 (Size: 30 MB)
First Fit
Free
10 MB
P1
40 MB
P3
100 MB
P2
8 MB
200 MB

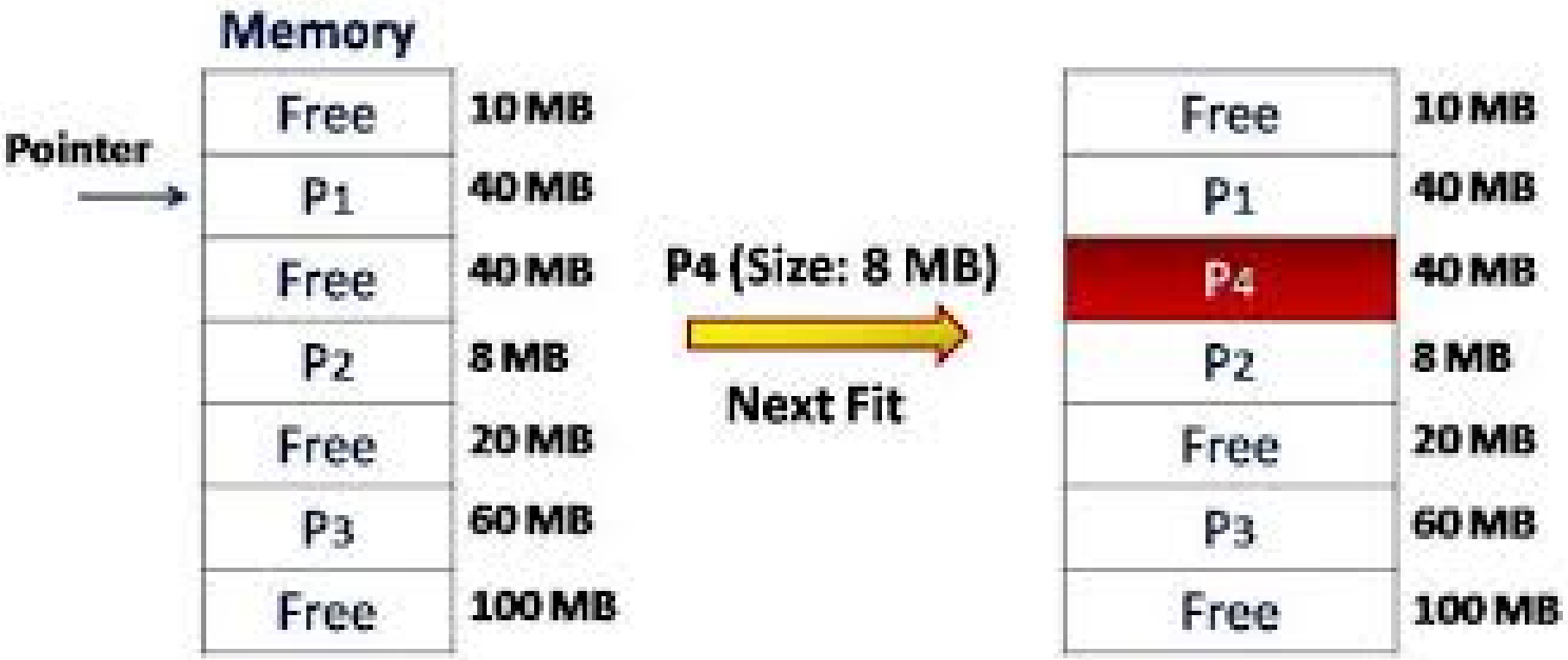
Best Fit


Memory
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
Free
100 MB
P4 (Size: 18 MB)
Best Fit
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
P4
20 MB
P3
60 MB
100 MB

Worst Fit


Memory
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
Free
100 MB
P4 (Size: 18 MB)
Worst Fit
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
P4 (15 MB)
100 MB
Free
85 MB

Next Fit


Memory
Pointer
Free
10 MB
P1
40 MB
Free
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
Free
100 MB
P4 (Size: 8 MB)
Next Fit
Free
10 MB
P1
40 MB
P4
40 MB
P2
8 MB
Free
20 MB
P3
60 MB
Free
100 MB

First, Best, Worst Fits