Singly Linked Lists

Subhabrata Samajder

IIIT, Delhi

Winter Semester,
24th March, 2023

Linked List

Problems With Arrays

- Many applications require that the number of elements changes dynamically as the algorithm progresses.

- **One possibility:** Define all the elements as arrays (or records) large enough to ensure sufficient storage space.
    - Often a good solution.
    - But requires storage according to the worst case (which may be unknown) which may be *inefficient*.
    - **Reason:** Array size has to be fixed in the beginning.

Problems With Arrays

- May need to perform insertions and deletions in the middle.
  - Need to shift all other elements.
  - Therefore very costly for large arrays - $\mathcal{O}(n)$.
- This problem is inherent to consecutive representation of arrays (or records).
- For these cases we need **dynamic data structures**.

Handling Lists

- Consider a list of integers

$$\{16, 8, 10, 2, 34, 20, 12, 32, 18, 9, 3\}$$

- It can be thought of as an element 16 followed by another list

$$16 - -\{8, 10, 2, 34, 20, 12, 32, 18, 9, 3\}$$

- Next list can also be thought of as 8 followed by a list

$$16 - -8 - -\{10, 2, 34, 20, 12, 32, 18, 9, 3\}$$

- $\therefore$ any list can be thought of as an element followed by a list.
- This gives a recursive definition of a linked list.

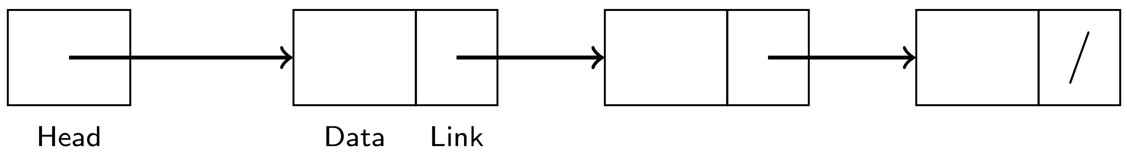
Linked Lists

- Linked lists are the simplest form of dynamic data structures.
- The objects are arranged in a linear order.
- Provides a simple, flexible representation for dynamic sets, supporting (though not necessarily efficiently) all the operations of a dynamic set.

List stored in an array $A[\,]$

Memory Address	**Array Index**	**List Contents**
3200	A[0]	36
3204	A[1]	42
3208	A[2]	20
3212	A[3]	16
3216	A[4]	38
3220	A[5]	40
3224	A[6]	12
3228	A[7]	54
3232	A[8]	82

Linked List



- A linked list is a series of connected nodes.
- Each node contains at least
  - A piece of data (any type)
  - Link to the next node in the list
- Head: points to the first node
- Links are generated by system.
- The last node points to **nil**.

List Items Stored In A Linked List

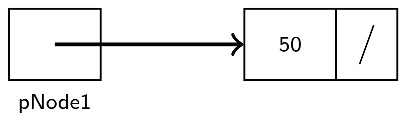
Memory Address	**Data Contents**	**Link Contents**
2020	36	450
450	42	3600
3600	20	4200
4200	16	4231
4231	38	760
760	40	5555
5555	12	**nil**

Defining a Node in C

```
typedef struct Node {
   int nData;
   struct Node *pNext;
} Node;

int main() {
   Node Node1, *pNode2;
   ⋮
```

Creating Two Nodes

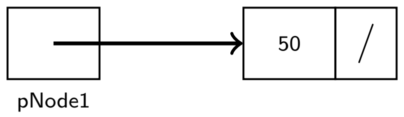
- Creating a node with value 50:



Creating Two Nodes

- Creating a node with value 50:

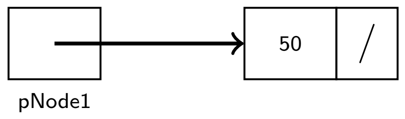
```
Node *pNode1 = NULL;

pNode1 = (Node *)malloc(sizeof(Node));
pNode1->nData = 50;

pNode1->pNext = NULL;
```



Creating Two Nodes

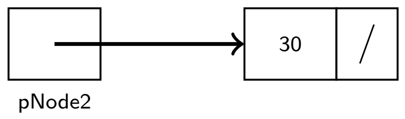
- Creating a node with value 50:

```
Node *pNode1 = NULL;

pNode1 = (Node *)malloc(sizeof(Node));
pNode1->nData = 50;
pNode1->pNext = NULL;
```

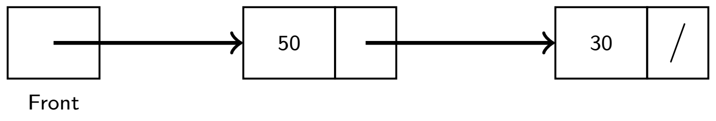


- Similarly creating a node with value 30:

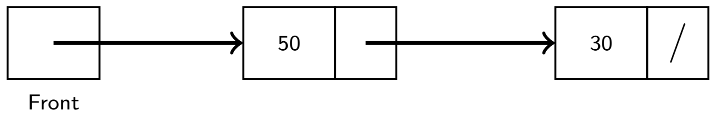
```
Node *pNode2 = NULL;

pNode2 = (Node *)malloc(sizeof(Node));
pNode2->nData = 30;
pNode2->pNext = NULL;
```



Linked List With First Node Followed By Second Node

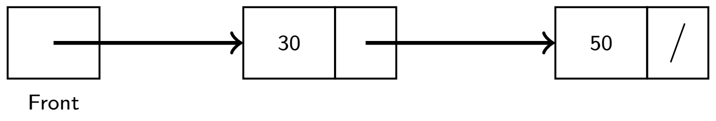


Linked List With First Node Followed By Second Node

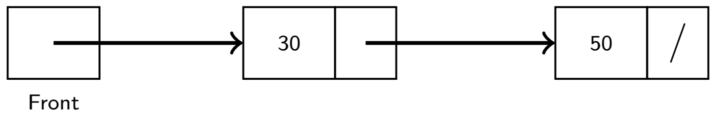


```
Node *pFront = NULL;

pFront = pNode1;
pFront->pNext = pNode2;
```

Linked List With Second Node Followed By First Node



Linked List With Second Node Followed By First Node



```
Node *pFront = NULL;

pFront = pNode2;
pFront->pNext = pNode1;
```

Inserting a Node in a Linked List

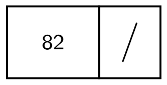
Inserting a Node at the Front of a List

Inserting a node containing value 82 at the front of the linked list.

Inserting a Node at the Front of a List

Inserting a node containing value 82 at the front of the linked list.

- First form a new node with value 82 and pointing to null.

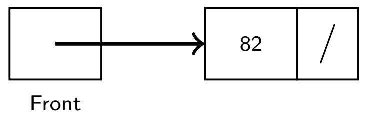


```
Node *pTemp = NULL;

pTemp = (Node *)malloc(sizeof(Node));
pTemp->nData = 82;

pTemp->pNext = Null;
```

Inserting a Node at the Front of a List

Inserting a node containing value 82 at the front of the linked list.

- First form a new node with value 82 and pointing to null.
- **Empty List:** The new node becomes the Front node.

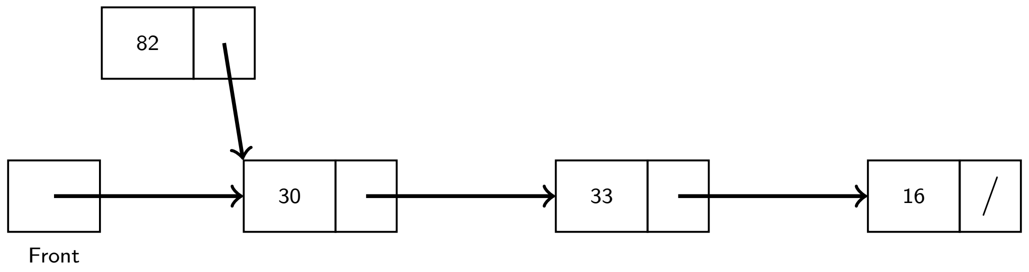


```
if (pFront == NULL)
  pFront = pTemp;
```

Inserting a Node at the Front of a List

Inserting a node containing value 82 at the front of the linked list.

- First form a new node with value 82 and pointing to null.
- **Otherwise:**
  - Link node 82 to Front (node containing 50).

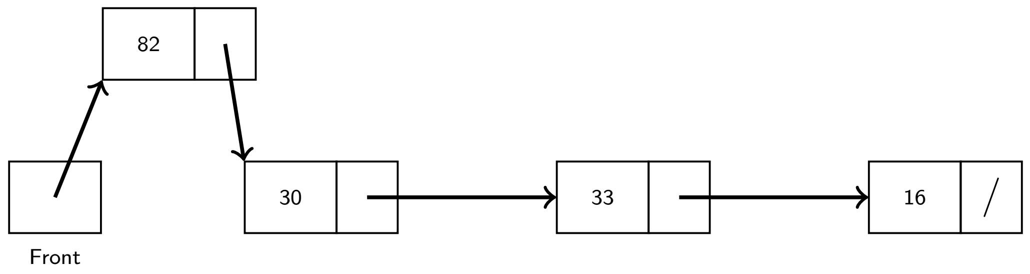


```
 if (pFront == NULL)
   pFront = pTemp;
else
   pTemp->pNext = pFront;
```

Inserting a Node at the Front of a List

Inserting a node containing value 82 at the front of the linked list.

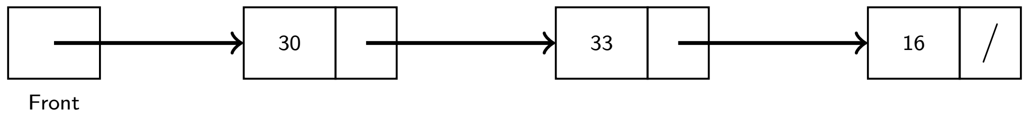
- First form a new node with value 82 and pointing to null.
- **Otherwise:**
  - Link node 82 to Front (node containing 50).
  - Finally, we declare node 82 to be the new Front node.



```
 if (pFront == NULL)
   pFront = pTemp;
else
   pTemp->pNext = pFront;

   pFront = pTemp;
```

Inserting a Node at Rear of a List

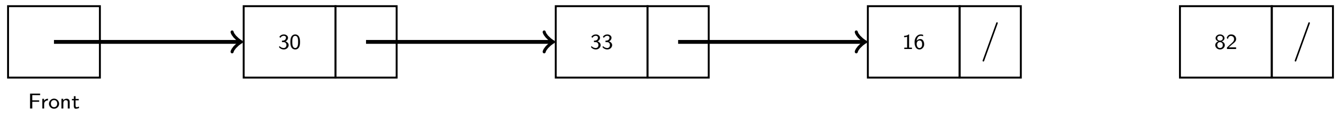
Insert a node containing the value 82 at the rear of the linked list.



Inserting a Node at Rear of a List

Insert a node containing the value 82 at the rear of the linked list.

- First form a new node with value 82 and pointing to null.

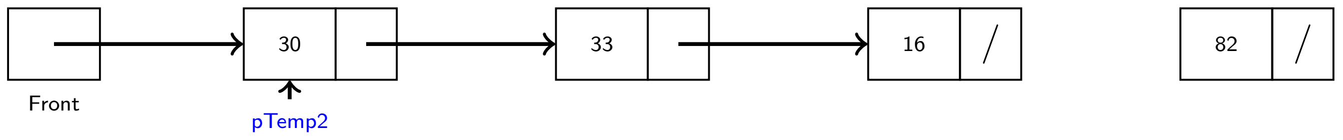


```
Node *pTemp1 = NULL, *pTemp2 = NULL;

pTemp1 = (Node *)malloc(sizeof(Node));
pTemp1->nData = 82;

pTemp1->pNext = Null;
```

Inserting a Node at Rear of a List

Insert a node containing the value 82 at the rear of the linked list.

- First form a new node with value 82 and pointing to null.
- **Empty List:** The new node becomes the Front node.
- Traverse through the list to reach the last node.

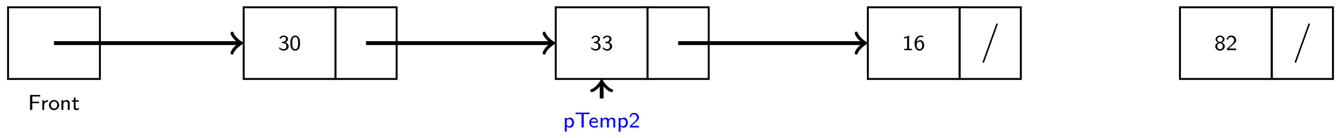


```
 if (pFront == NULL)
   pFront = pTemp1;
else {
   pTemp2 = pFront;
```

Inserting a Node at Rear of a List

Insert a node containing the value 82 at the rear of the linked list.

- First form a new node with value 82 and pointing to null.
- **Empty List:** The new node becomes the Front node.
- Traverse through the list to reach the last node.

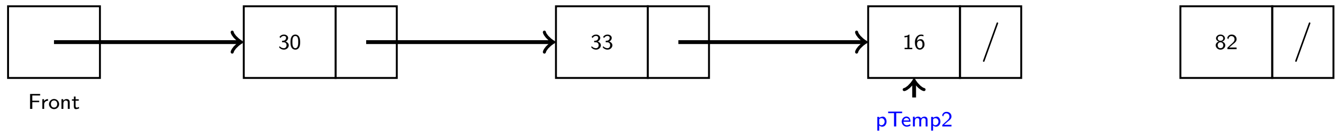


```
 if (pFront == NULL)
   pFront = pTemp1;
else {
   pTemp2 = pFront;

   while (pTemp2->pNext != NULL)
      pTemp2 = pTemp2->pNext;
```

Inserting a Node at Rear of a List

Insert a node containing the value 82 at the rear of the linked list.

- First form a new node with value 82 and pointing to null.
- **Empty List:** The new node becomes the Front node.
- Traverse through the list to reach the last node.

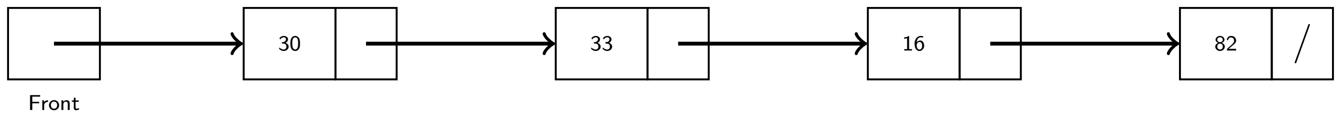


```
 if (pFront == NULL)
   pFront = pTemp1;
else {
   pTemp2 = pFront;

   while (pTemp2->pNext != NULL)
      pTemp2 = pTemp2->pNext;
```

Inserting a Node at Rear of a List

Insert a node containing the value 82 at the rear of the linked list.

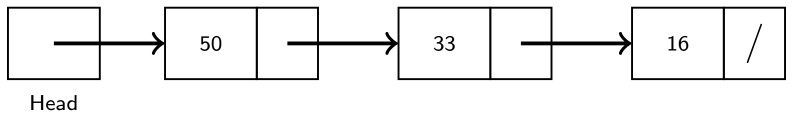
- First form a new node with value 82 and pointing to null.
- **Empty List:** The new node becomes the Front node.
- Traverse through the list to reach the last node.
- Point the last node to the newly formed node with value 82.



```
 if (pFront == NULL)
   pFront = pTemp1;
else {
   pTemp2 = pFront;

   while (pTemp2->pNext != NULL)
      pTemp2 = pTemp2->pNext;

   pTemp2->pNext = pTemp1; }
```

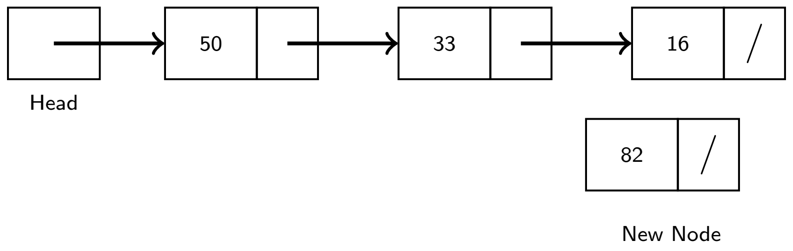
Inserting a Node at a Specific Position

- Insert a node containing value 82 at position 3 after 33.
- Involves breaking the list and setting two links.



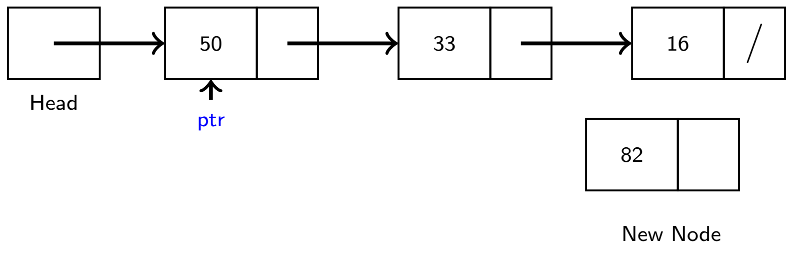
Inserting a Node at a Specific Position

- Insert a node containing value 82 at position 3 after 33.
- Involves breaking the list and setting two links.
- First form a new node with value 82 (and pointing to null).



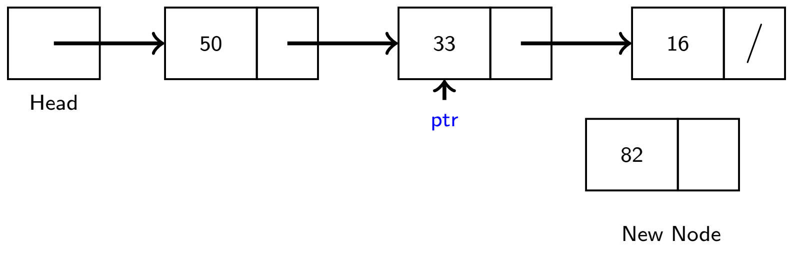
Inserting a Node at a Specific Position

- Insert a node containing value 82 at position 3 after 33.
- Involves breaking the list and setting two links.
- First form a new node with value 82 (and pointing to null).
- Find the node you want to insert after.



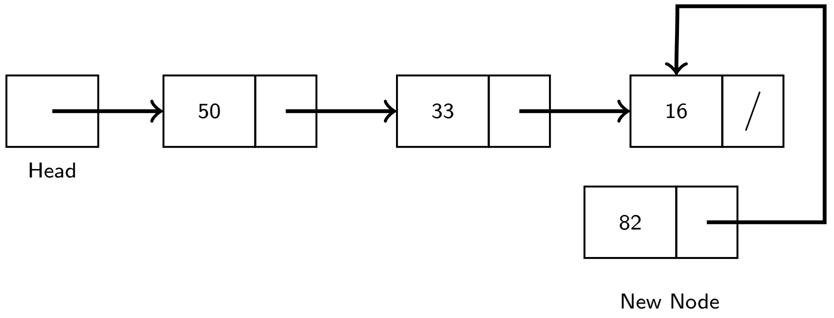
Inserting a Node at a Specific Position

- Insert a node containing value 82 at position 3 after 33.
- Involves breaking the list and setting two links.
- First form a new node with value 82 (and pointing to null).
- Find the node you want to insert after.



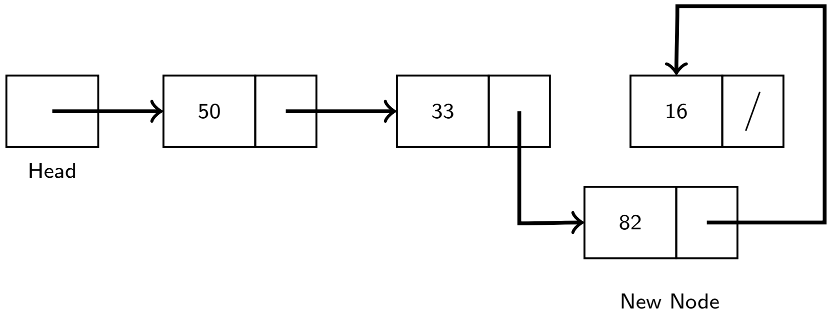
Inserting a Node at a Specific Position

- Insert a node containing value 82 at position 3 after 33.
- Involves breaking the list and setting two links.
- First form a new node with value 82 (and pointing to null).
- Find the node you want to insert after.
- Copy the link from the node that's already in the list.



Inserting a Node at a Specific Position

- Insert a node containing value 82 at position 3 after 33.
- Involves breaking the list and setting two links.
- First form a new node with value 82 (and pointing to null).
- Find the node you want to insert after.
- Copy the link from the node that's already in the list.
- Change the link in the node that's already in the list.

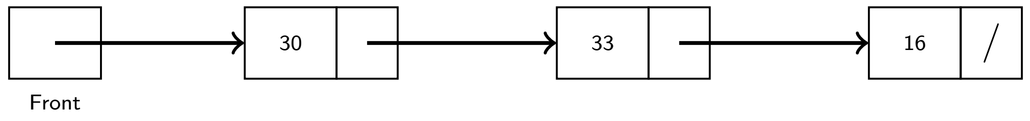


Inserting a Node at a Specific Position in the List: C

```
void insertAtPos(int val , int pos, Node *pFront) {
  Node *pTemp = NULL;
  int nIter = 1;    // Position of first node is assumed 1.

  pTemp = (Node *)malloc(sizeof(Node));
  pTemp->nData = val;
  pTemp->pNext = NULL;

  while (pFront != NULL) {
    if (nIter == pos) {
      if (pFront->pNext == NULL) {
        pFront->pNext = pTemp;
        break;
      }
      else {
        pTemp->pNext = pFront->pNext;
        pFront->pNext = pTemp;
        break;
      }
    }
    pFront = pFront->pNext;
    nIter++;
  }
}
```

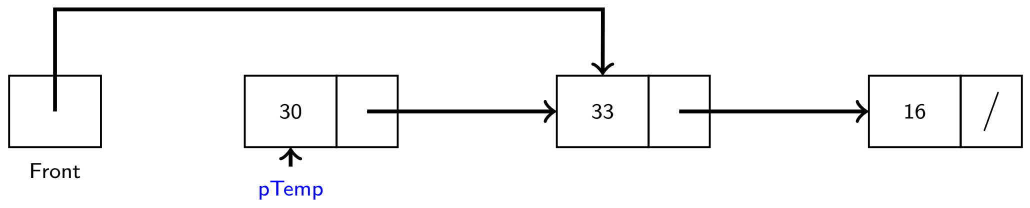
Deletion in Singly Linked Lists

Deleting the First Node



Deleting the First Node

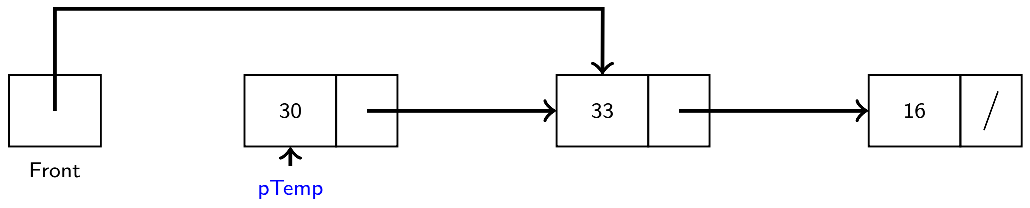
- To delete the first element, change the link in the header.



```
Node *pTemp = NULL;

if (pFront == NULL)
   return 0;
else {
   pTemp = pFront;

   pFront = pFront->pNext;
```

Deleting the First Node

- To delete the first element, change the link in the header.

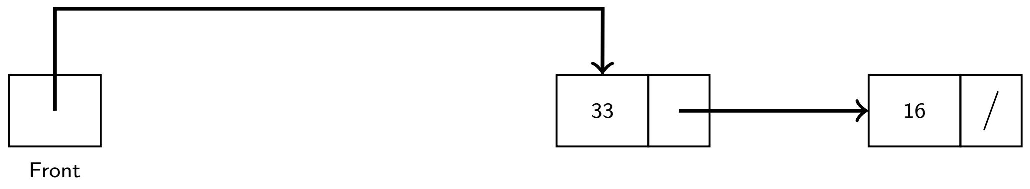


```
Node *pTemp = NULL;

if (pFront == NULL)
   return 0;
else {
   pTemp = pFront;

   pFront = pFront->pNext;
```

What about node 30?

Deleting the First Node

- To delete the first element, change the link in the header.



```
Node *pTemp = NULL;

if (pFront == NULL)
   return 0;
else {
   pTemp = pFront;
   pFront = pFront->pNext;

   free(pTemp);
}
```

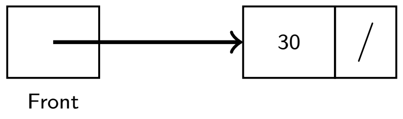
Deleting the Last Node

- **Empty List:** Nothing to do!

```
if (pFront == NULL)
   return 0;
```

Deleting the Last Node

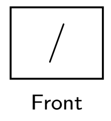
- **Empty List:** Nothing to do!
- **List of size 1:**



```
if (pFront == NULL)
   return 0;
if (pFront->pNext == NULL) {
```

Deleting the Last Node

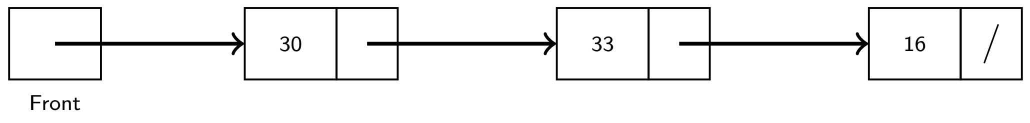
- **Empty List:** Nothing to do!
- **List of size 1:** Set Front = null.



```
if (pFront == NULL)
   return 0;
if (pFront->pNext == NULL) {
   pTemp = pFront;
   pFront = NULL;
   free(pTemp);

   return 0;
}
```

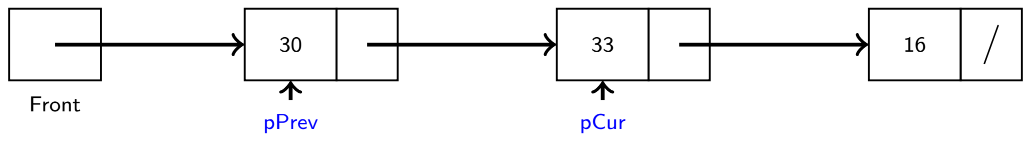
Deleting the Last Node

- Set pointer pPrev to first and pCur to second node of the list.



Deleting the Last Node

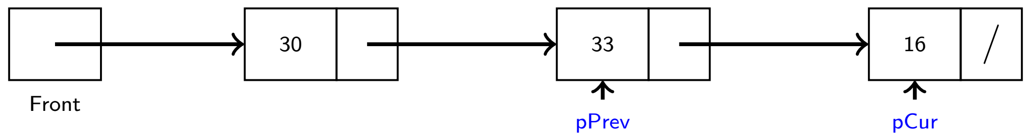
- Set pointer pPrev to first and pCur to second node of the list.



```
  Node *pPrev = NULL, *pCur = NULL;

pPrev = pFront;
pCur = pFront->pNext;
```

Deleting the Last Node

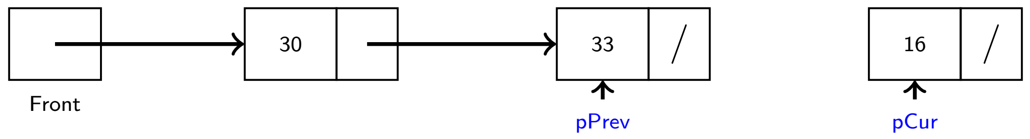
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse till pCur->pNext == NULL.



```
 Node *pPrev = NULL, *pCur = NULL;

pPrev = pFront;
pCur = pFront->pNext;
while (pCur->pNext != NULL) {
   pPrev = pCur;
   pCur = pCur->pNext; }
```

Deleting the Last Node

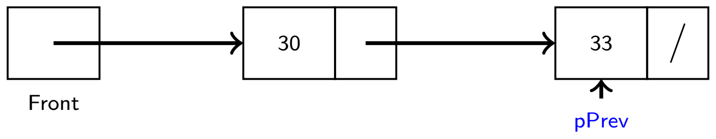
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse till pCur->pNext == NULL.
- Set link for pPrev to null, so that last node is not reachable.



```
 Node *pPrev = NULL, *pCur = NULL;

pPrev = pFront;
pCur = pFront->pNext;
while (pCur->pNext != NULL) {
   pPrev = pCur;
   pCur = pCur->pNext; }
pPrev->pNext = NULL;
```

Deleting the Last Node

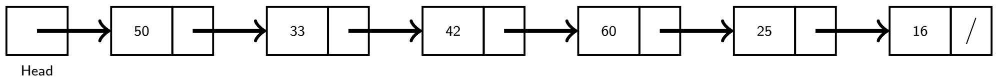
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse till pCur->pNext == NULL.
- Set link for pPrev to null, so that last node is not reachable.
- **To free memory in C:** free(pCur).



```
  Node *pPrev = NULL, *pCur = NULL;

pPrev = pFront;
pCur = pFront->pNext;
while (pCur->pNext != NULL) {
   pPrev = pCur;
   pCur = pCur->pNext; }
pPrev->pNext = NULL;
free(pCur);
```

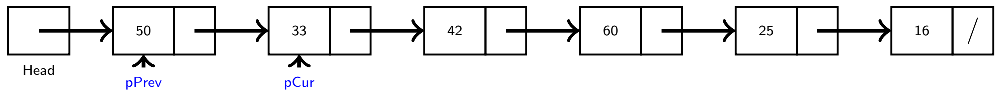
Deleting a Node Containing Data d

- Assume that the list is of length at least 2.
- Let $d = 60$.



Deleting a Node Containing Data d

- Assume that the list is of length at least 2.
- Let $d = 60$.
- Set pointer pPrev to first and pCur to second node of the list.

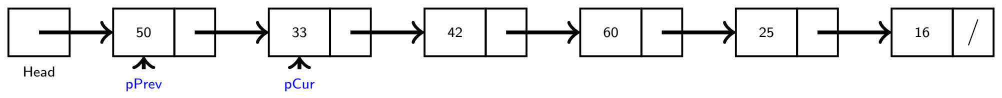


pPrev = pFront;

pCur = pFront->pNext;

Deleting a Node Containing Data d

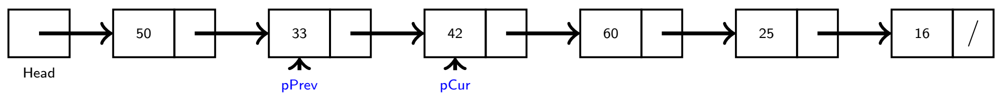
- Assume that the list is of length at least 2.
- Let $d = 60$.
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse until pCur->nData == 60.



```
pPrev = pFront;
pCur = pFront->pNext;
while ( pCur->nData != d ) {
   pPrev = pCur;
   pCur = pCur->pNext; }
```

Deleting a Node Containing Data *d*

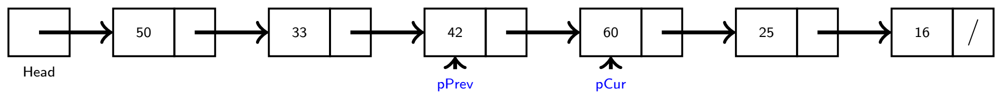
- Assume that the list is of length at least 2.
- Let $d = 60$.
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse until pCur->nData == 60.



```
pPrev = pFront;
pCur = pFront->pNext;

while ( pCur->nData != d ) {
   pPrev = pCur;
   pCur = pCur->pNext; }
```

Deleting a Node Containing Data d

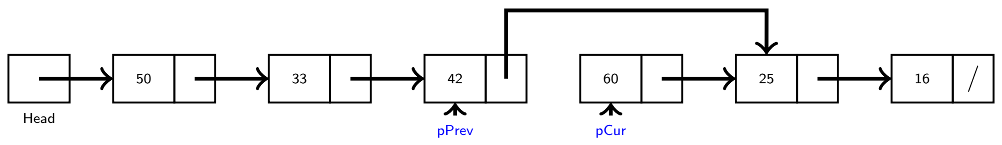
- Assume that the list is of length at least 2.
- Let $d = 60$.
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse until pCur->nData == 60.



```
pPrev = pFront;
pCur = pFront->pNext;
while ( pCur->nData != d ) {
   pPrev = pCur;
   pCur = pCur->pNext; }
```

Deleting a Node Containing Data *d*

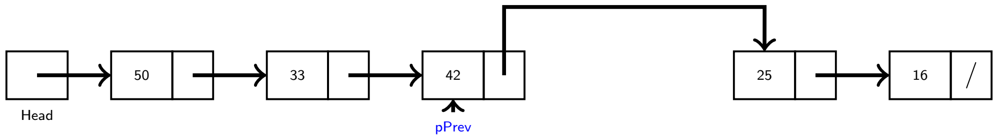
- Assume that the list is of length at least 2.
- Let $d = 60$.
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse until pCur->nData == 60.
- Set link of pPrev to next node of pCur.



```
pPrev = pFront;
pCur = pFront->pNext;

while ( pCur->nData != d ) {
   pPrev = pCur;
   pCur = pCur->pNext; }

pPrev->pNext = pCur->pNext;
```

Deleting a Node Containing Data d

- Assume that the list is of length at least 2.
- Let $d = 60$.
- Set pointer pPrev to first and pCur to second node of the list.
- Traverse until pCur->nData == 60.
- Set link of pPrev to next node of pCur.



```
pPrev = pFront;
pCur = pFront->pNext;
while ( pCur->nData != d ) {
   pPrev = pCur;
   pCur = pCur->pNext; }
pPrev->pNext = pCur->pNext;

free(pCur);
```

Books Consulted

1. Chapter 10.2 of *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.

Thank You for your kind attention!

Questions!!