

# Foundational Data Structures

# Arrays

- Fixed, Contiguous Blocks of Memory of the same type
- Pros
  - Random Access
- Cons
  - Cannot Resize
  - Not great if the size is not known or fixed

Array of Size 10 In Memory

Identifier	Contents	Byte Address
...	...	...
a[]	36	28
...	....	...
a[0]	256	36
a[1]		42
a[2]		48
...		
a[9]	NULL	90

# Growable Arrays

- Array Lists
  - Strings
- Growing
  - Create a new array with a larger size
  - Transfer all the data from the original array to the new array
  - Remove the original Array
- Pros
  - Semi-Random Access
  - Growable Structure
- Cons
  - Lots of Overhead
  - Does not Shrink
  - Not Great for Large Amounts of Data
  - Not Great Performance

## Growing Concept

Full Array		Larger Array	
Index	Values	Index	Values
0	1	0	1
1	2	1	2
2	4	2	4
3	4	3	4
4	5	4	5
5	6	5	6
6	7	6	7
		7	0
		8	0
		9	0

# IS THERE A BETTER WAY!?



# Linking Structures

# Linking Structures

- Groups Together
  - Data
  - Link(s) / Reference(s) / Pointer(s)
  - “Node”
- Pros
  - Growable
  - Shrinkable
- Cons
  - No Random Access

Node



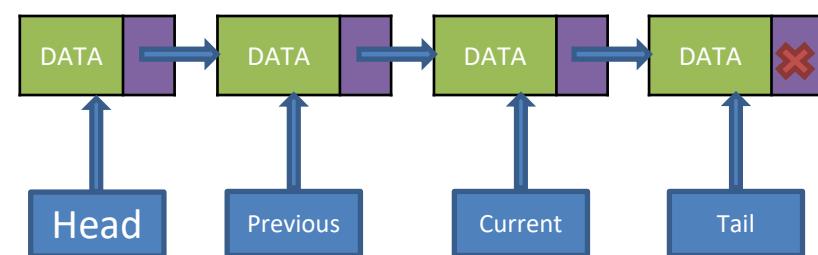
List of Nodes



# Linked Lists

- Nodes Contain
  - Data
  - Link
- Special Nodes
  - Head: Always points to the first element of the list
  - Tail: Always points to the last element of the list
  - Current: Movable pointer used to Access and Modify Data in the List
  - Previous: Always stays on node behind Current
- Certain Linked Lists may omit some of these Nodes

Linked List



# Linked List Set Up

# Internal Classes

- Class within Classes
- Aids in grouping together like-information that is only used within a class
- Other Programmers do not need access to these classes

## Syntax

```
public <<class identifier>>
{
    private <<internal class identifier>>
    {
        //Body of Internal Class
    }
}
```

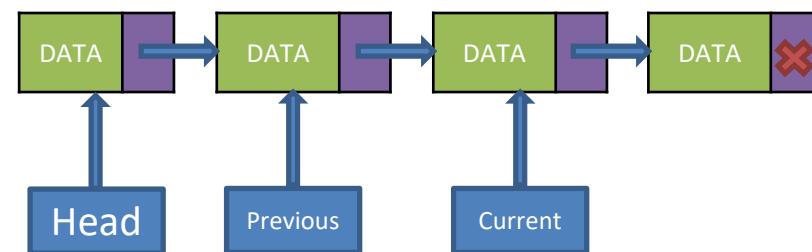
## Example

```
public class IntLL
{
    private class ListNode
    {
    }
}
```

# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

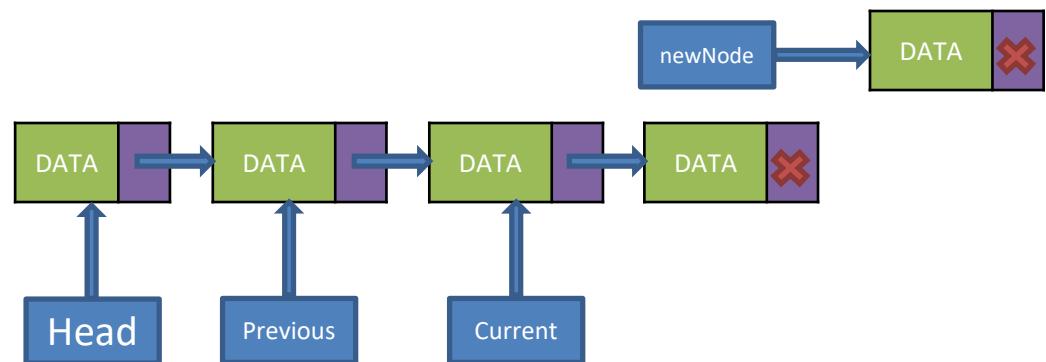
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

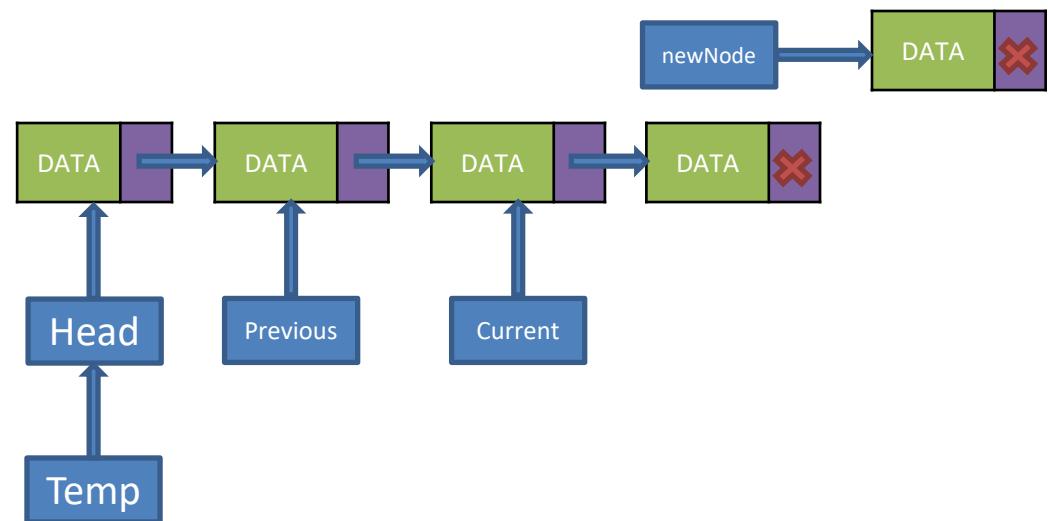
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

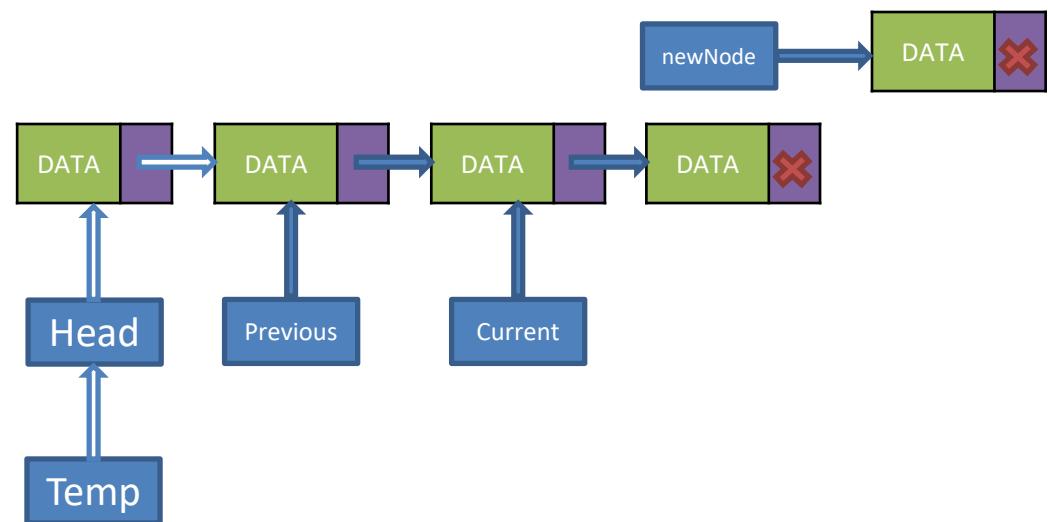
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

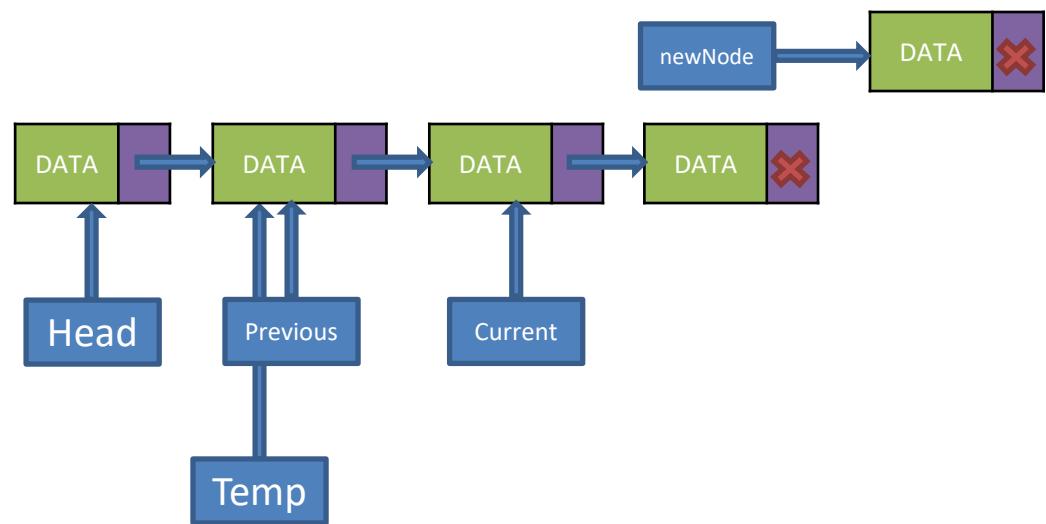
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

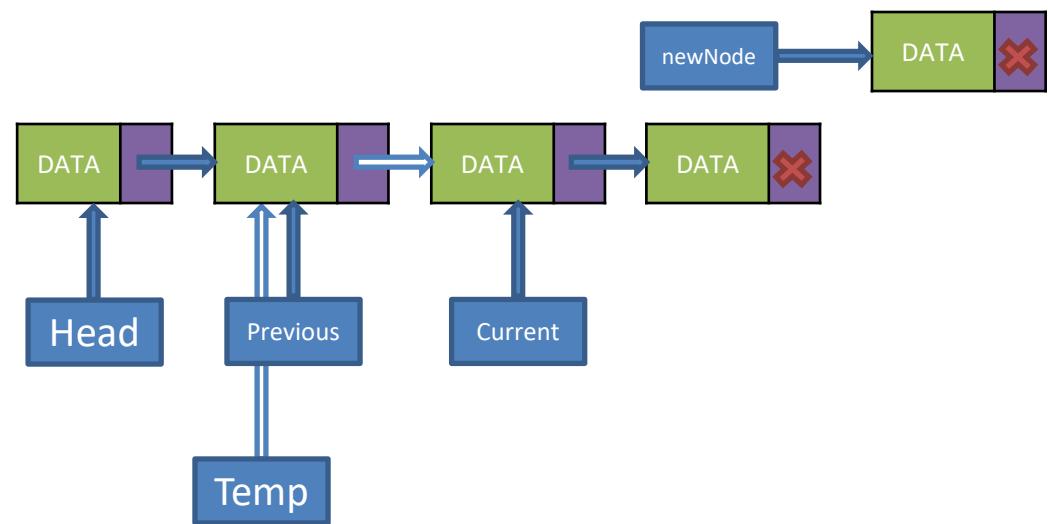
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

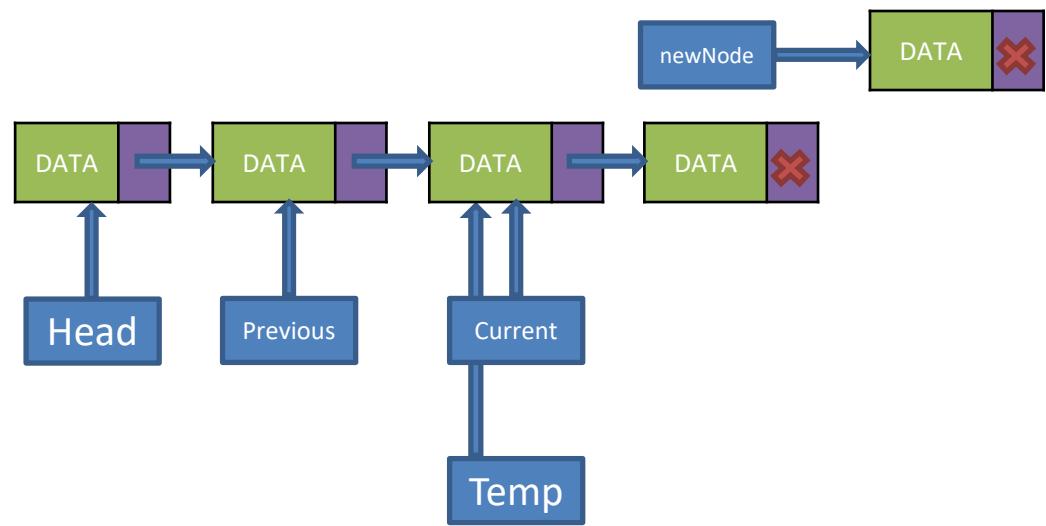
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

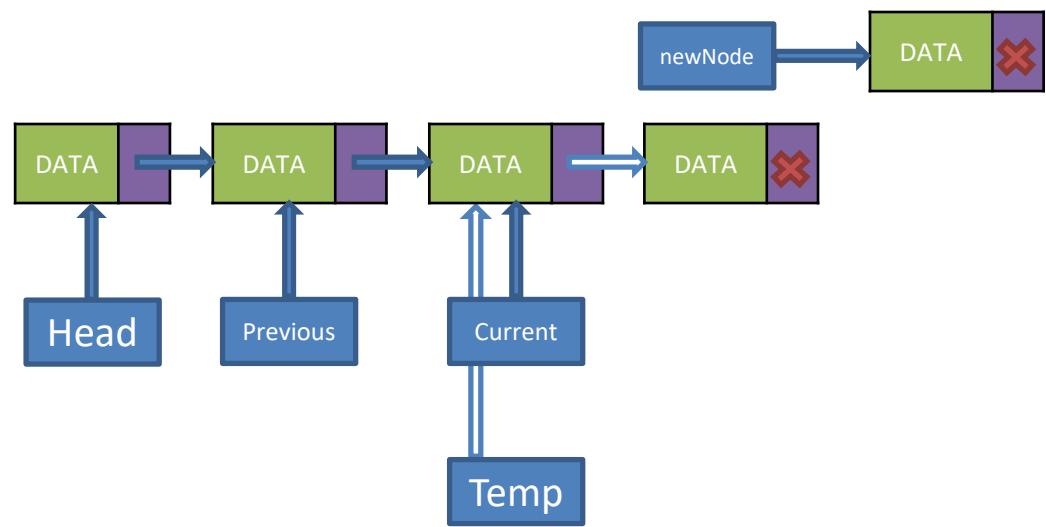
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

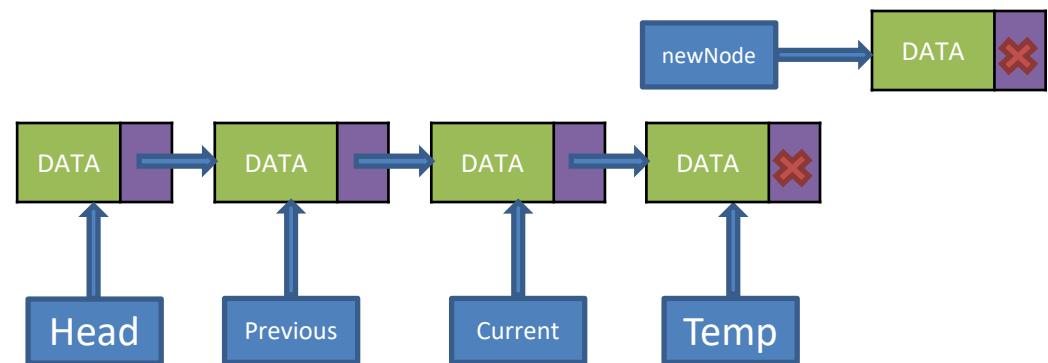
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

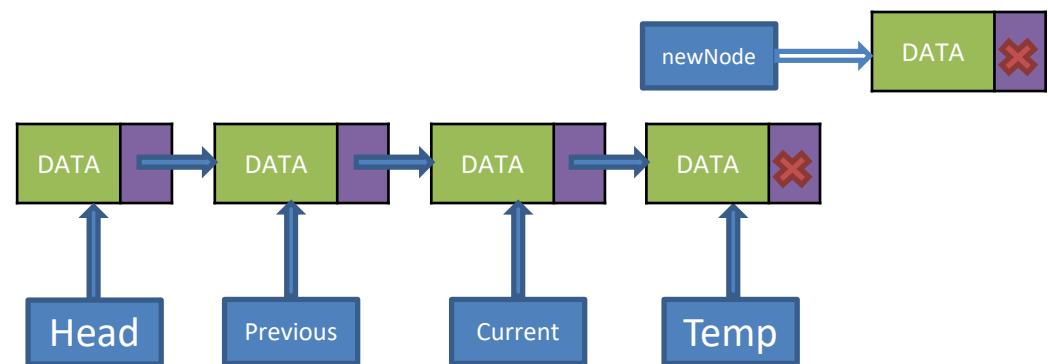
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

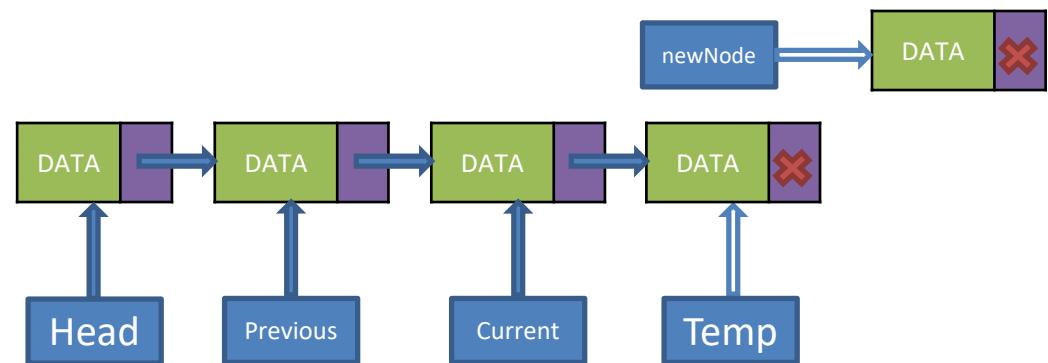
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

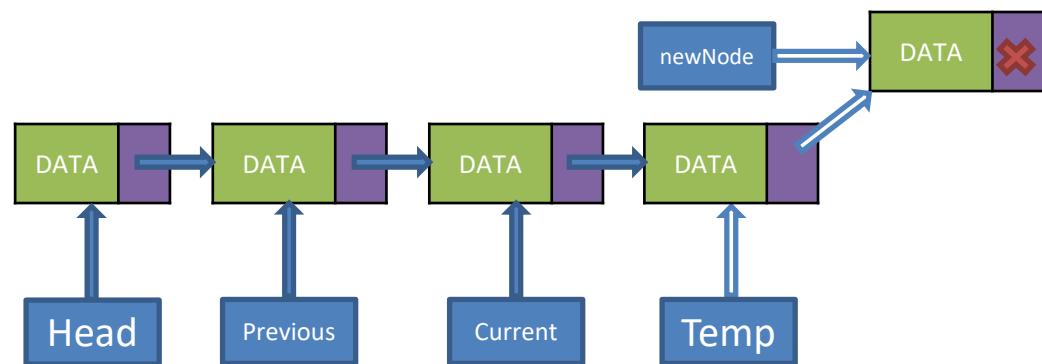
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

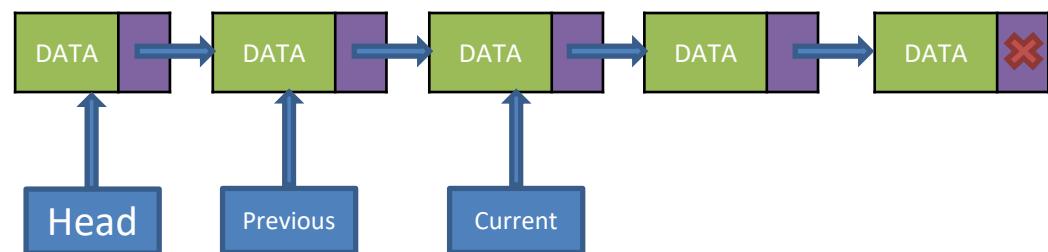
## Concept



# Adding

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

## Concept



# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory			
Identifier	Contents	Byte Address	
...	...	...	...
iLinkedList	NULL	28	
...	....	...	...
IntLL	-	50	
head	NULL	64	
current	NULL	70	
previous	NULL	76	
...	...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
...	...	...
aData	4	96
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

4

Memory		
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
...	...	...
aData	4	96
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

4

Memory			
Identifier	Contents	Byte Address	
...	...	...	...
iLinkedList	50	28	
...	...	...	...
IntLL	-	50	
head	NULL	64	
current	NULL	70	
previous	NULL	76	
...	...	...	...
aData	4	96	
...	...	...	...
newNode	NULL	104	
...	...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

4

Memory		
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
...	...	...
aData	4	96
...	...	...
newNode	NULL	104
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory		
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
...	...	...
aData	4	96
...	...	...
newNode	128	104
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

4

Memory		
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
...	...	...
aData	4	90
...	...	...
newNode	128	104
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

4

Memory		
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
...	...	...
aData	4	90
...	...	...
newNode	128	104
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

4

Memory		
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	NULL	64
current	128	70
previous	NULL	76
...	...	...
aData	4	90
...	...	...
newNode	128	104
...	...	...

# Adding in Memory

```

    4
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
...

```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	NULL	134
head	128	64	...	...	...
current	128	70	...	...	...
previous	NULL	76	...	...	...
...	...	...	...	...	...
aData	4	00	...	...	...
...	...	...	...	...	...
newNode	128	104	...	...	...
...	...	...	...	...	...

# Adding in Memory

```

    4
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
...

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	4	96
...	...	...
newNode	128	104
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...

# Adding in Memory

```

    4
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
...

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	4	96
...	...	...
newNode	128	104
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...

# Adding in Memory

```

    4
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...

# Adding in Memory

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

# Adding in Memory

```
>public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...

# Adding in Memory

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

## More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...

# Adding in Memory

3

```

public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

```

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

## More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...

# Adding in Memory

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

## More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...

# Adding in Memory

```

    public void add(int aData)
    {
        ListNode newNode = new ListNode(aData,null);
        if(head == null)
        {
            head = current = newNode;
            return;
        }
        ListNode temp = head;
        while(temp.link != null)
        {
            temp = temp.link;
        }
        temp.link = newNode;
    }

```

3

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

## More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...
temp	NULL	355
...	...	...

# Adding in Memory

```

    public void add(int aData)
    {
        ListNode newNode = new ListNode(aData,null);
        if(head == null)
        {
            head = current = newNode;
            return;
        }
        ListNode temp = head;
        while(temp.link != null)
        {
            temp = temp.link;
        }
        temp.link = newNode;
    }
}

```

3

## Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

## More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...
temp	128	355
...	...	...

# Adding in Memory

```

    3
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
...

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...
temp	128	355
...	...	...

# Adding in Memory

```

    3
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

```

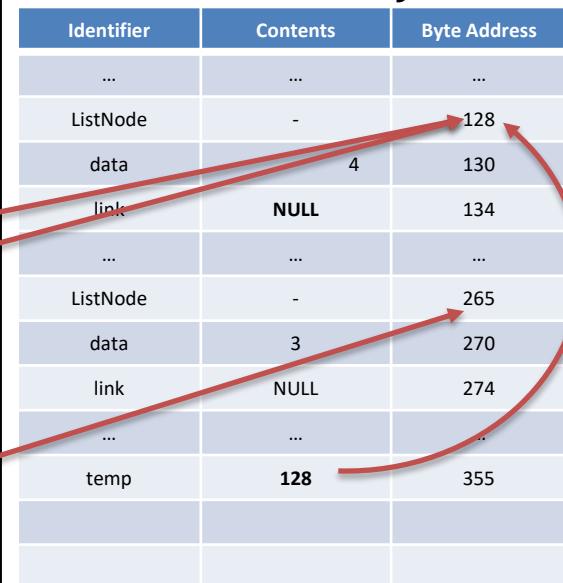


Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	NULL	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...
temp	128	355
...	...	...



# Adding in Memory

```

    3
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...
temp	128	355
...	...	...

# Adding in Memory

```

    3
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...
aData	3	96
...	...	...
newNode	265	104
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...
temp	128	355
...	...	...

# Adding in Memory

```

    3
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

```



Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...

# Adding in Memory

```

    public void add(int aData)
    {
        ListNode newNode = new ListNode(aData,null);
        if(head == null)
        {
            head = current = newNode;
            return;
        }
        ListNode temp = head;
        while(temp.link != null)
        {
            temp = temp.link;
        }
        temp.link = newNode;
    }

```

2

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	128	70
previous	NULL	76
...	...	...

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode	-	265
data	3	270
link	NULL	274
...	...	...

# Adding in Memory

```

    3
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}

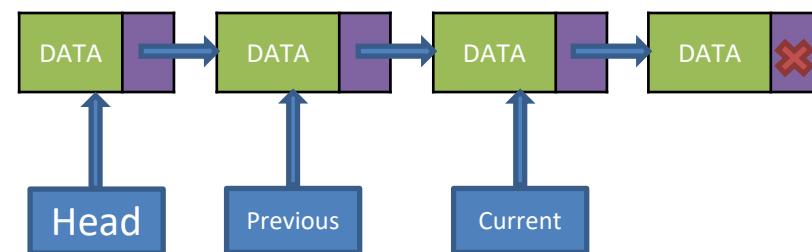
```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	265	134
head	128	64	...	...	...
current	128	70	ListNode	-	265
previous	NULL	76	data	3	270
...	...	...	link	374	274
			...	...	...
			ListNode	-	374
			data	2	380
			link	NULL	384

# Adding After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

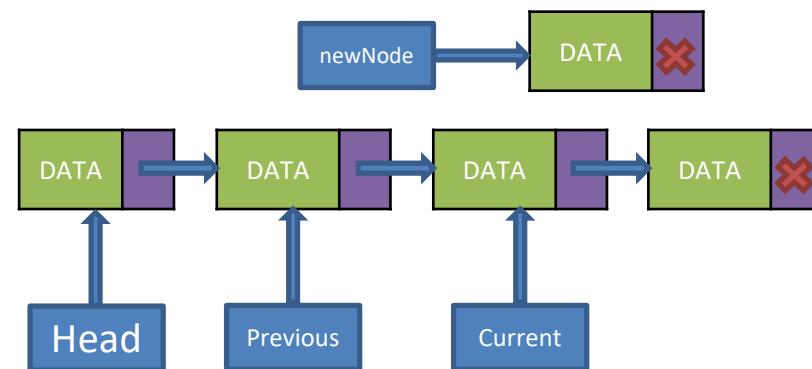
## Concept



# Adding After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

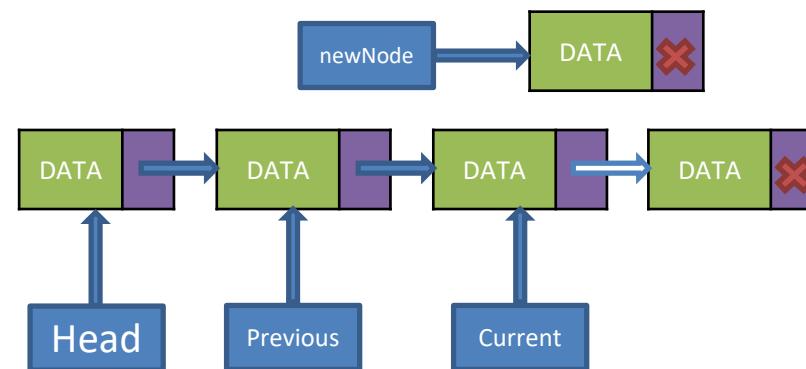
## Concept



# Adding After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

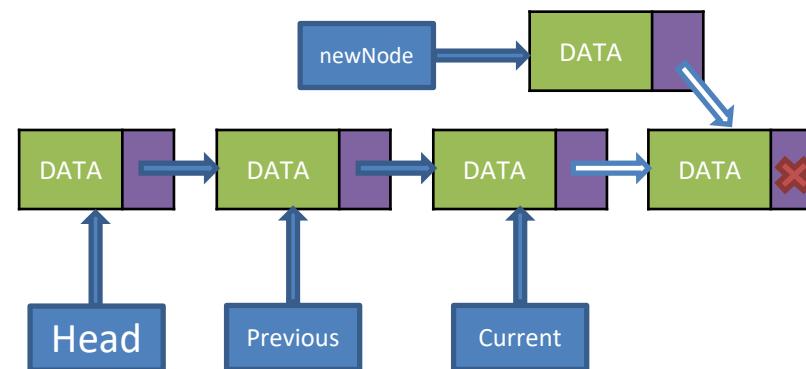
## Concept



# Adding After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

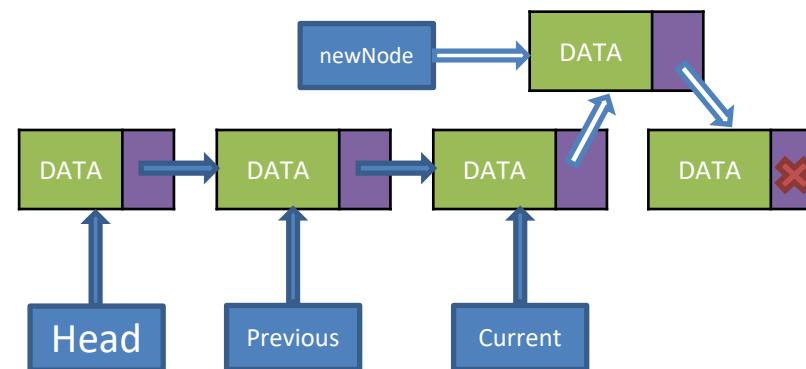
## Concept



# Adding After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

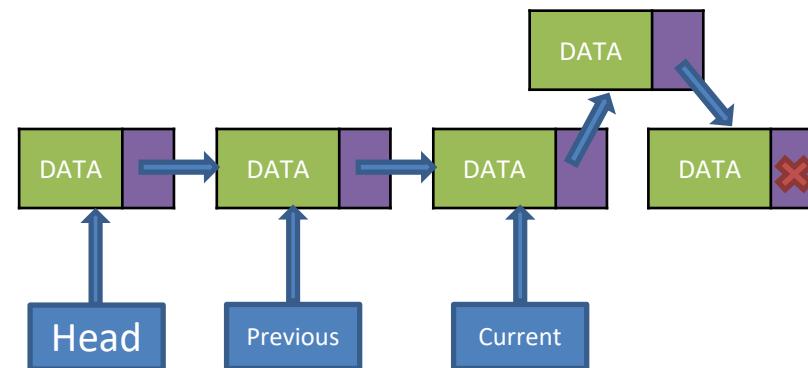
## Concept



# Adding After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

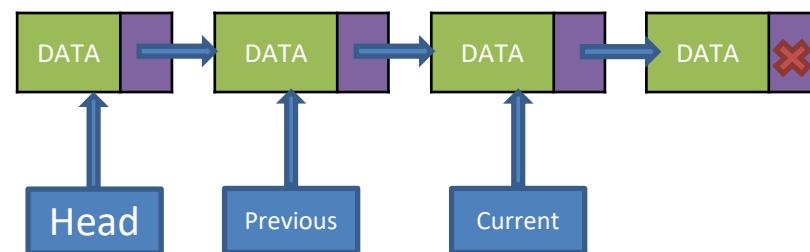
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

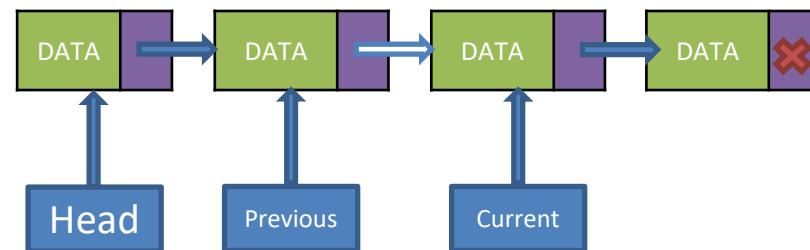
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

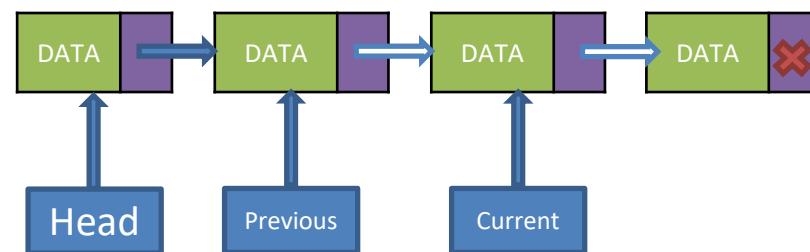
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

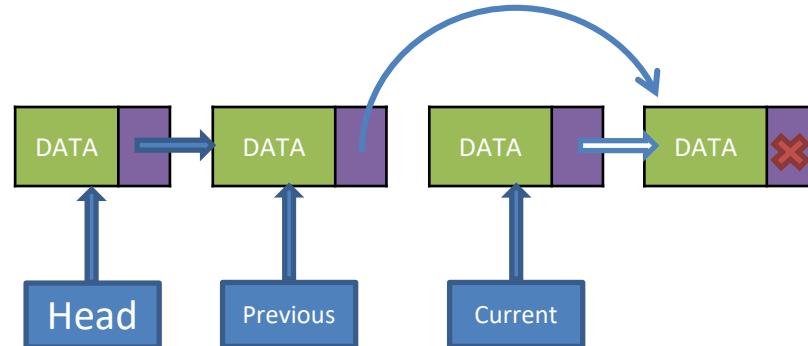
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

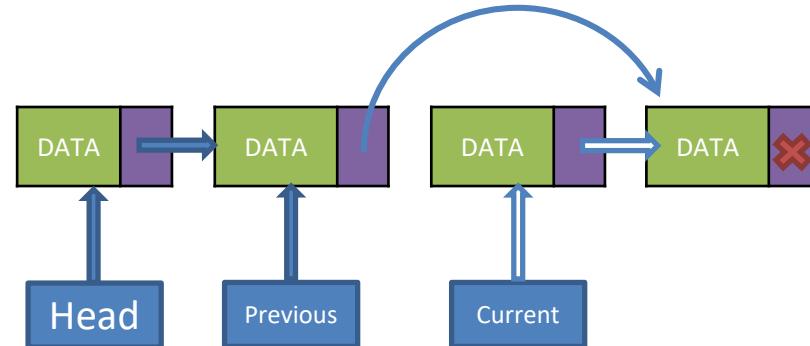
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

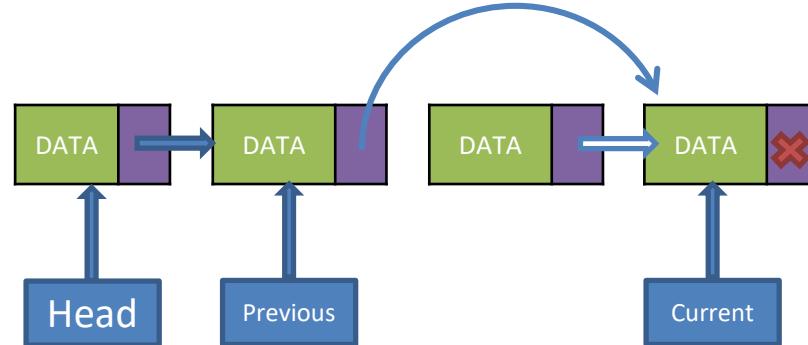
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

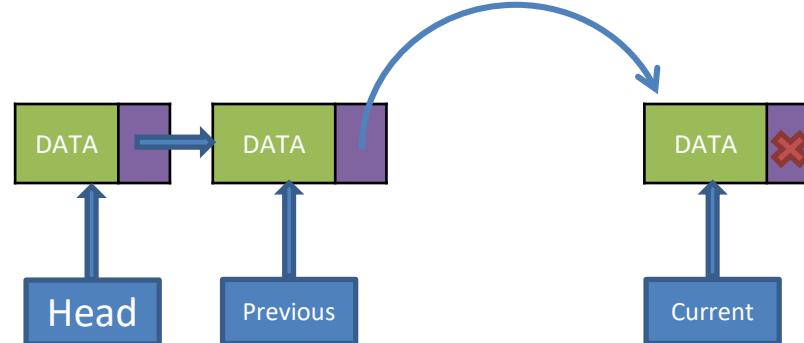
## Concept



# Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous's Link to Current's Link
- Move Current Forward

Concept



# Removing Current in Memory

```

public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    { //At head node
        head = current.link;
        current = head;
    }
}

```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	265	134
head	128	64	...	...	...
current	265	70	ListNode	-	265
previous	128	76	data	3	270
...	...	...	link	374	274
			...	...	...
			ListNode	-	374
			data	2	380
			link	NULL	384

The diagram illustrates the state of memory before and after the removal of a node from a linked list. In the 'Memory' section, several pointers are modified: 'current' points to the next node (265), 'previous' points to the previous node (128), and 'IntLL' points to the removed node (50). In the 'More Memory' section, the freed memory block (265) is shown, consisting of a ListNode structure with data value 3 and link value 374.

# Removing Current in Memory

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}
```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	265	134
head	128	64	...	...	...
current	265	70	ListNode	-	265
previous	128	76	data	3	270
...	...	...	link	374	274
			...	...	...
			ListNode	-	374
			data	2	380
			link	NULL	384

The diagram illustrates the state of memory before and after the execution of the `removeCurrent()` method. In the 'Memory' section, several pointers are being updated:

- `previous.link` is set to `current.link`, indicated by a blue arrow.
- `current` is set to `current.link`, indicated by a purple arrow.
- `head` is updated to point to the new head node (`current`), indicated by a purple arrow.
- `current` is also updated to point to the previous node (`previous`), indicated by an orange arrow.
- `previous` is updated to point to the previous node (`head`), indicated by a blue arrow.

In the 'More Memory' section, the freed memory block is shown. It contains a `ListNode` structure with a `data` field of value 3 and a `link` field of address 374. This freed block is then reused to create a new node with `data` 2 and `link` NULL, starting at address 380.

# Removing Current in Memory

```

public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}

```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	265	70
previous	128	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode	-	265
data	3	270
link	374	274
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

# Removing Current in Memory

```

public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    { //At head node
        head = current.link;
        current = head;
    }
}

```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	265	134
head	128	64	...	...	...
current	265	70	ListNode	-	265
previous	128	76	data	3	270
...	...	...	link	374	274
			...	...	...
			ListNode	-	374
			data	2	380
			link	NULL	384

The diagram illustrates the state of memory before and after the removal of a node from a linked list. In the 'Memory' section, several pointers are modified: 'previous.link' is set to 'current.link', 'current' is set to 'current.link', and 'head' is set to 'current'. In the 'More Memory' section, the freed memory block is shown, consisting of a ListNode node with data value 3 and link value 374.

# Removing Current in Memory

```

public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    { //At head node
        head = current.link;
        current = head;
    }
}

```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	374	134
head	128	64	...	...	...
current	265	70	ListNode	-	265
previous	128	76	data	3	270
...	...	...	link	374	274
			...	...	...
			ListNode	-	374
			data	2	380
			link	NULL	384

The diagram illustrates the state of memory before and after the removal of a node from a linked list. In the 'Memory' section, several pointers are being modified:

- The pointer 'previous.link' is updated to point to the node's link (50), indicated by a red arrow.
- The pointer 'current' is moved to point to the next node (70), indicated by a blue arrow.
- The pointer 'head' is updated to point to the new head node (265), indicated by a purple arrow.
- The pointer 'previous' remains at 128, indicated by an orange arrow.

In the 'More Memory' section, the freed memory block is shown:

- A red arrow points from the original address 374 to the freed block.
- A blue arrow points from the original address 128 to the freed block.
- An orange arrow points from the original address 265 to the freed block.
- The freed block contains a 'data' field with value 3 and a 'link' field with value 374.

# Removing Current in Memory

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    { //At head node
        head = current.link;
        current = head;
    }
}
```

Memory			More Memory		
Identifier	Contents	Byte Address	Identifier	Contents	Byte Address
...	...	...	...	...	...
iLinkedList	50	28	ListNode	-	128
...	...	...	data	4	130
IntLL	-	50	link	374	134
head	128	64	...	...	...
current	265	70	ListNode	-	265
previous	128	76	data	3	270
...	...	...	link	374	274
			...	...	...
			ListNode	-	374
			data	2	380
			link	NULL	384

The diagram illustrates the state of memory before and after the removal of a node from a linked list. In the 'Memory' section, the 'current' pointer is moved to point to the next node (address 70), and the 'previous' pointer's link is updated to point to the previous node (address 76). In the 'More Memory' section, the freed memory block at address 265 is shown, consisting of a ListNode structure with data value 3 and a link value of 374. After the removal, the 'link' field of the previous node (address 76) now points to this freed block (address 265), and the 'link' field of the freed block (address 265) is set to NULL.

# Removing Current in Memory

```

public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    { //At head node
        head = current.link;
        current = head;
    }
}

```

The diagram illustrates the state of memory before and after the execution of the `removeCurrent()` method.

**Memory State:**

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	...	...
IntLL	-	50
head	128	64
current	374	70
previous	128	76
...	...	...

**More Memory:**

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	374	134
...	...	...
ListNode	-	265
data	3	270
link	374	274
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

Annotations show the movement of pointers and the deallocation of memory:

- A blue arrow points to the `current` pointer in the first table, which is then shown pointing to the `data` field of a `ListNode` object in the second table.
- An orange arrow points to the `previous` pointer in the first table, which is then shown pointing to the `link` field of the same `ListNode` object.
- A red arrow points to the `IntLL` pointer in the first table, which is then shown pointing to the `link` field of the `ListNode` object at address 374.
- Red arrows also point to the deallocation of the original `current` node (address 374) and its copy (address 128).

# Removing Current in Memory

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    { //At head node
        head = current.link;
        current = head;
    }
}
```

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	374	134
...	...	...
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

# Problems

- This is only a Linked List of Integers
- How can we make this same structure without having to rewrite the code for every type?

# Linking Structures