DATA STRUCTURES AND ITS APPLICATIONS

Vandana M L

Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Memory Allocation

Vandana M L

Department of Computer Science and Engineering

Memory Allocation

- Static Memory Allocation
- Dynamic Memory Allocation

Static Memory Allocation

- allocated by the compiler.
- Exact size and type of memory must be known at compile time
- Memory is allocated in stack area

```
int b;

int c[10] ;
```

Disadvantages of Static Memory Allocation

- Memory allocated can not be altered during run time as it is allocated during compile time
- This may lead to under utilization or over utilization of memory
- Memory can not be deleted explicitly only contents can be overwritten
- Useful only when data size is fixed and known before processing

Dynamic Memory Allocation

- Dynamic memory allocation is used to obtain and release memory during program execution.
- It operates at a low-level
- Memory Management functions are used for allocating and deallocating memory during execution of program
- These functions are defined in “stdlib.h”

Dynamic Memory Allocation Functions:

- Allocate memory - malloc(), calloc(), and realloc()
- Free memory - free()

Dynamic Memory Allocation Functions: malloc()

To allocate memory use

```
void *malloc(size_t size);
```

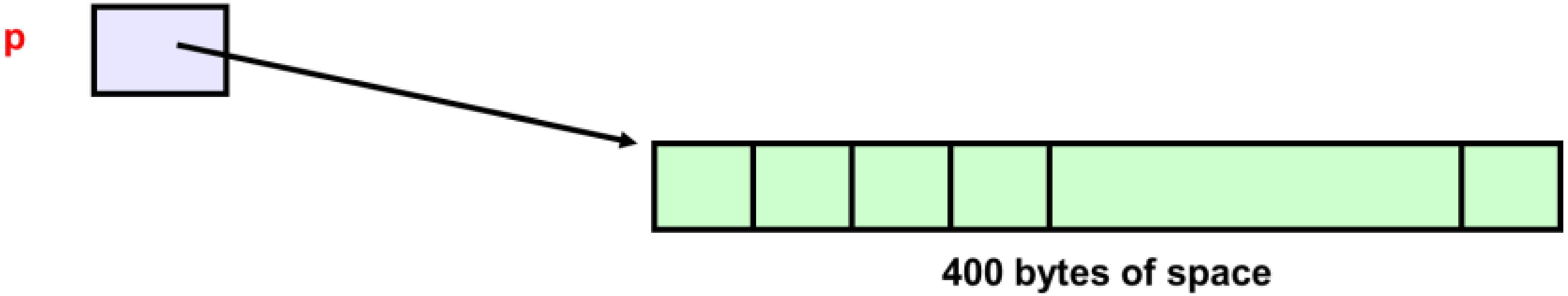
- Takes number of bytes to allocate as argument.
- Use sizeof to determine the size of a type.
- Returns pointer of type void *. A void pointer may be assigned to any pointer.
- If no memory available, returns NULL.

Dynamic Memory Allocation Functions: malloc()

To allocate space for 100 integers:

```
int *p;
if ((p = (int*)malloc(100 * sizeof(int))) == NULL){
   printf("out of memory\n");
   exit();
   }
```

- Note we cast the return value to int*.
- Note we also check if the function returns NULL.



Dynamic Memory Allocation Functions: malloc()

- cptr = (char *) malloc (20);

  Allocates 20 bytes of space for the pointer cptr of type char

- sptr = (struct stud *) malloc(10*sizeof(struct stud));

  Allocates space for a structure array of 10 elements. sptr points to a structure element of type struct stud

Always use sizeof operator to find number of bytes for a data type, as it can vary from machine to machine

Dynamic Memory Allocation Functions: malloc()

- malloc always allocates a block of contiguous bytes
  - The allocation can fail if sufficient contiguous memory space is not available
  - If it fails, malloc returns NULL

```
if  ((p = (int *) malloc(100 * sizeof(int))) == NULL)
 {
      printf ("\n Memory cannot be allocated");
      exit();
  }
```

The n integers allocated can be accessed as *p, *(p+1), *(p+2),..., *(p+n-1) or just as p[0], p[1], p[2], ...,p[n-1]

Dynamic Memory Allocation Functions: free()

To release allocated memory use

```
free(ptrvariable)
```

- Deallocates memory allocated by malloc().
- Takes a pointer as an argument.

Dangling pointer

e.g.

```
free(newPtr);
```

Memory Leak

Dynamic Memory Allocation Functions: calloc()

Similar to malloc(),

But allocated memory space are zero by default..

calloc() requires two arguments –

```
void *calloc(size_t nitem, size_t size);
```

Example

```
int *p;
p=(int*)calloc(100,sizeof(int));
```

returns a void pointer if the memory allocation is successful, else it'll return a NULL pointer.

Dynamic Memory Allocation Functions: realloc()

Reallocate a block

Two arguments

- Pointer to the already allocated block
- Size of new block

```
int *ip;
ip = (int*)malloc(100 * sizeof(int));
. . .
/*  need twice as much space */
ip = (int*)realloc(ip, 200 * sizeof(int));
```

Lecture Summary

Memory Allocation

- Static Memory allocation
- Dynamic memory allocation

Apply the concepts to implement C program for the following problem statement

- Multiply two matrices . Allocate the memory for the matrices dynamically

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

+91 7411716615

DATA STRUCTURES AND ITS APPLICATIONS

Vandana M L

Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Introduction to Singly Linked List

Vandana M L
Department of Computer Science and Engineering

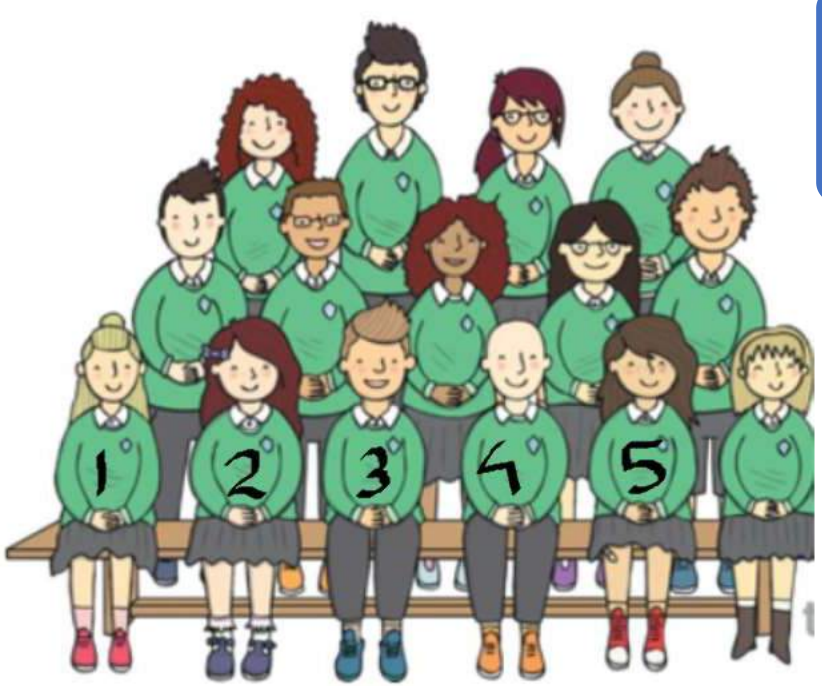
List

- Dynamic data structure consists of a collection of elements
- Can be implemented in two ways
  - By contiguous memory allocation : ArrayList
  - By Linked Allocation : Linked List

List Data Structure: Operations

- Creating a List
- Inserting an element in a list
- Deleting an element from a list
- Searching a list
- Reversing a list
- Concatenating two lists
- Traversing a list

Understanding Array List (Linear List using Arrays)

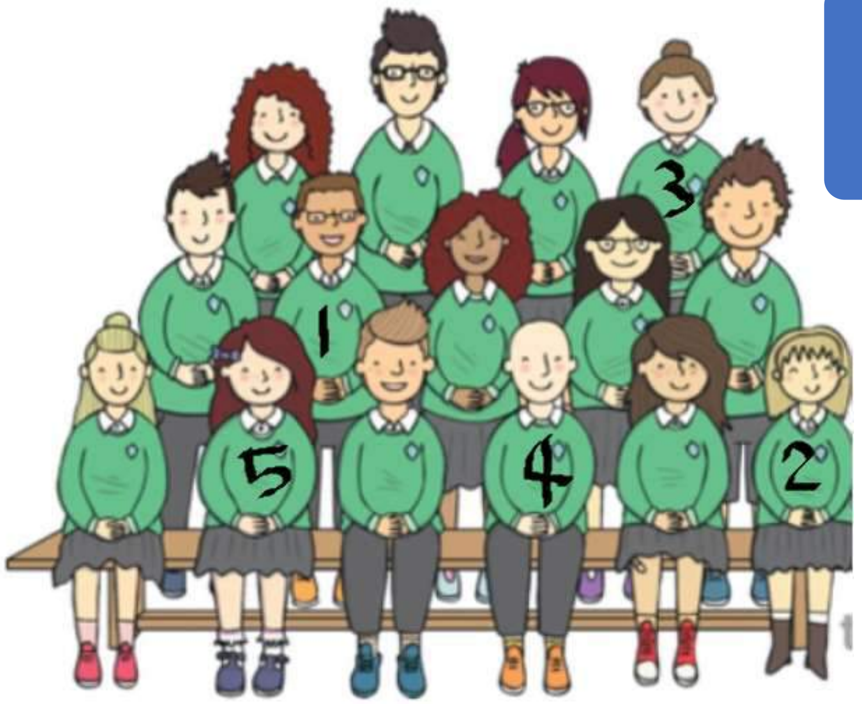


Placement
- Sequential

Access
- Sequential

Array List

Understanding Linked List



Placement
- Random

Access
- Sequential

Linked List

Array List Vs Linked List

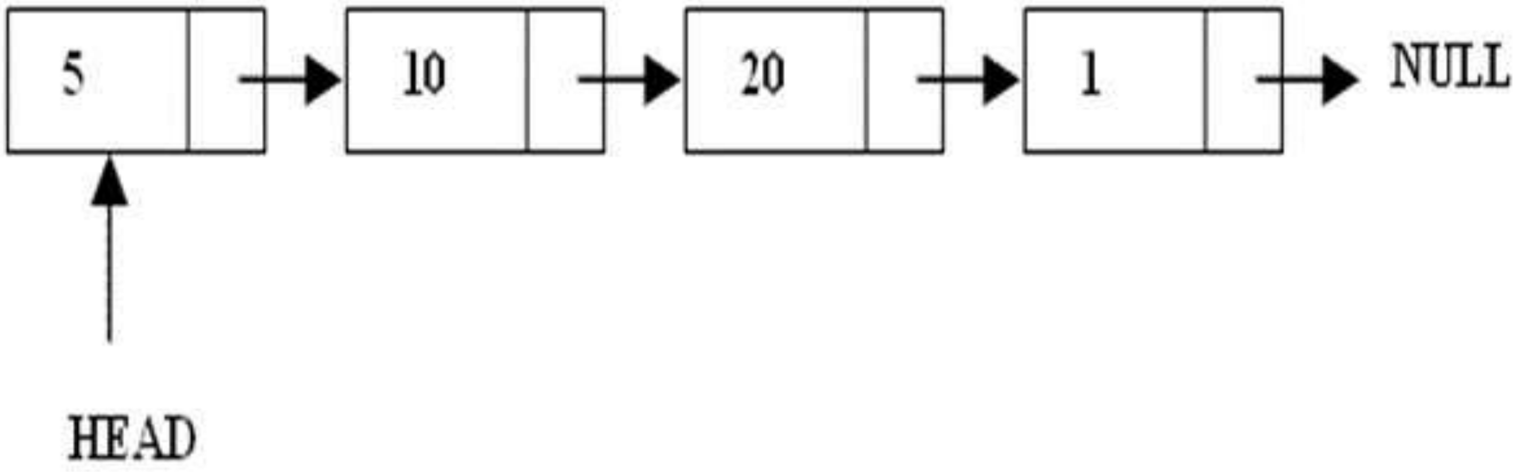
ArrayList	Linked list
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient	Insertions and Deletions are efficient
Elements in contiguous memory locations	Elements not in contiguous memory locations
May result in memory wasteage if all the allocated space is not used	Since memory is allocated dynamically(as per requirement) there is no wastage of memory.
Sequential and random access is faster	Sequential and random access is slow

Types of Linked List

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Multi Linked List

Singly Linked List

- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called as link.

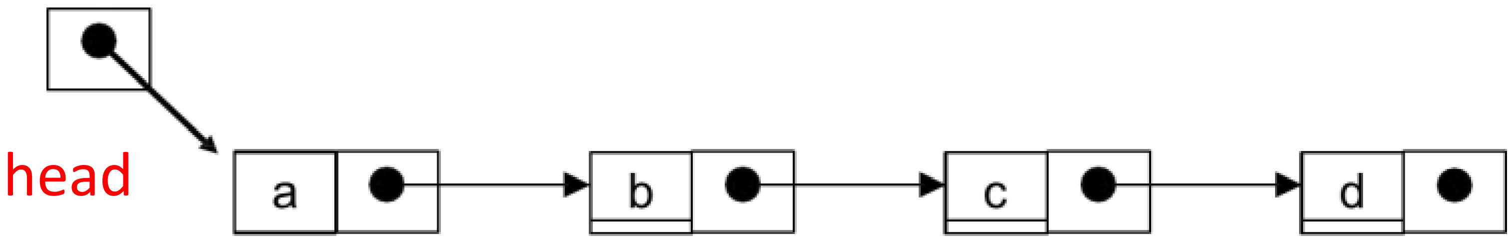


Single Linked List

- Each node has only one link part
- Each link part contains the address of the next node in the list
- Link part of the last node contains NULL value which signifies the end of the node

DATA	LINK

Single Linked List :Schematic representation



- Each node contains a value(data) and a pointer to the next node in the list

- Head/start is a pointer which points at the first node in the list

Singly Linked List

Apply the concepts to answer the following questions

- Give structure definition for node of singly linked list used to store employee data (employee no , name, salary ,designation)

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

+91 7411716615

DATA STRUCTURES AND ITS APPLICATIONS

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Singly Linked List

Vandana M L

Department of Computer Science and Engineering

Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

Deleting first node

Case 1: Linked list is empty

Case 2: Linked list with a single node

- delete the node
- set head to NULL

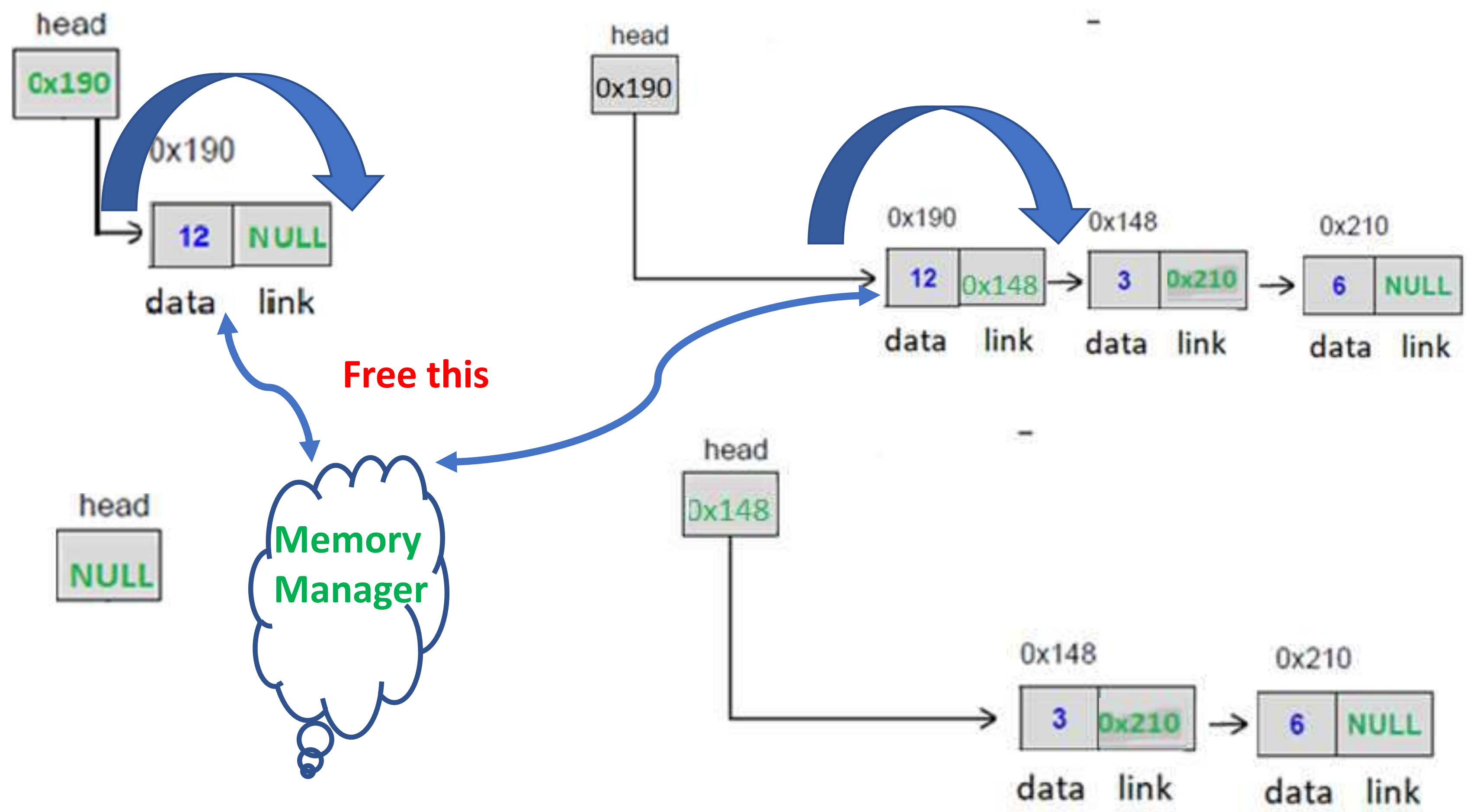
Map to one case

Case3:Linked list has more than one node

- Change head to point to second node
- Delete the first node

Deleting first node

Only one node in list **More than one node**



Deleting last node

Case 1: Linked list is empty

Case 2: Linked list with a single node

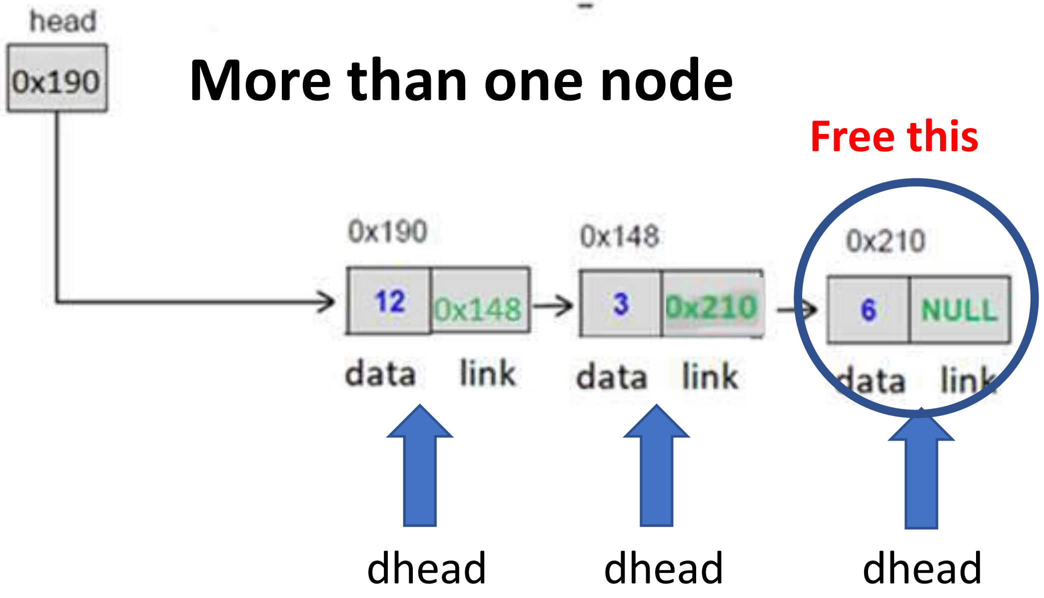
- delete the node
- set head to NULL

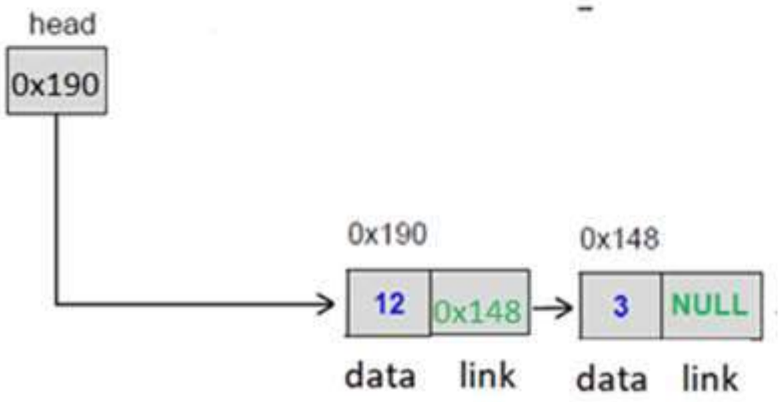
Case3:Linked list has more than one node

- Traverse the linked list to point to second last node
- Delete the last node
- Set link field of second last node to NULL

Deleting last node

More than one node





Deleting node from a given position

If the linked list is not empty

If position is 1

- Delete from the front of the linked list

Else

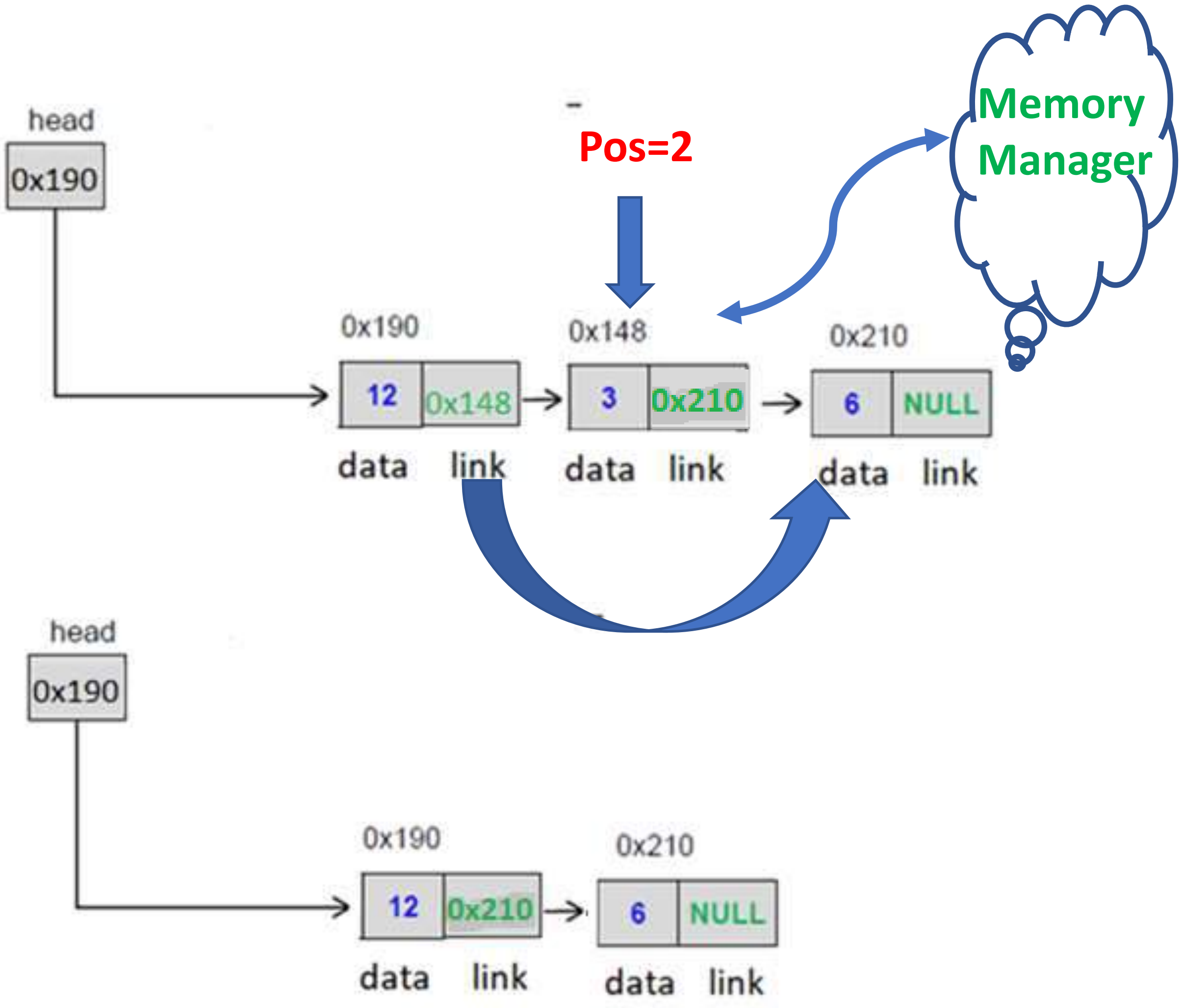
If position is a valid position

- Traverse linked list to get the desired position
- keep track of previous node
- set previous node link field to link field of current node
- delete the current node

Else

- print invalid position

Deleting node from a given position



Singly Linked List delete operation

Apply the concepts to implement following operations for a singly linked list

- Delete a node with given key value
- Delete all alternate nodes
- Delete all the nodes (erase the linked list)

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

+91 7411716615

DATA STRUCTURES AND APPLICATIONS

Vandana M L

Department of Computer Science and Engineering

DATA STRUCTURES AND APPLICATIONS

Doubly Linked List

Vandana M L

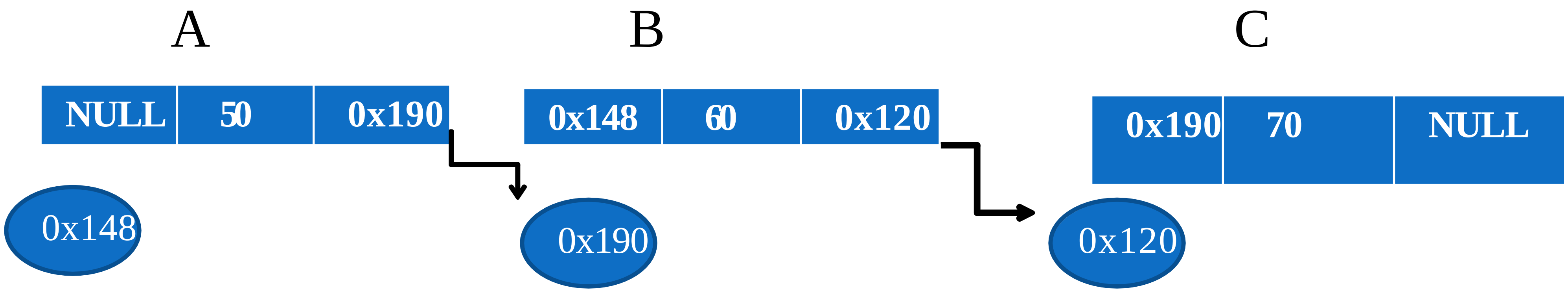
Department of Computer Science and Engineering

A doubly linked list contain three fields:

- Data
- link to the next node
- link to the previous node.

Doubly Linked List :Node Structure

previous	data	next

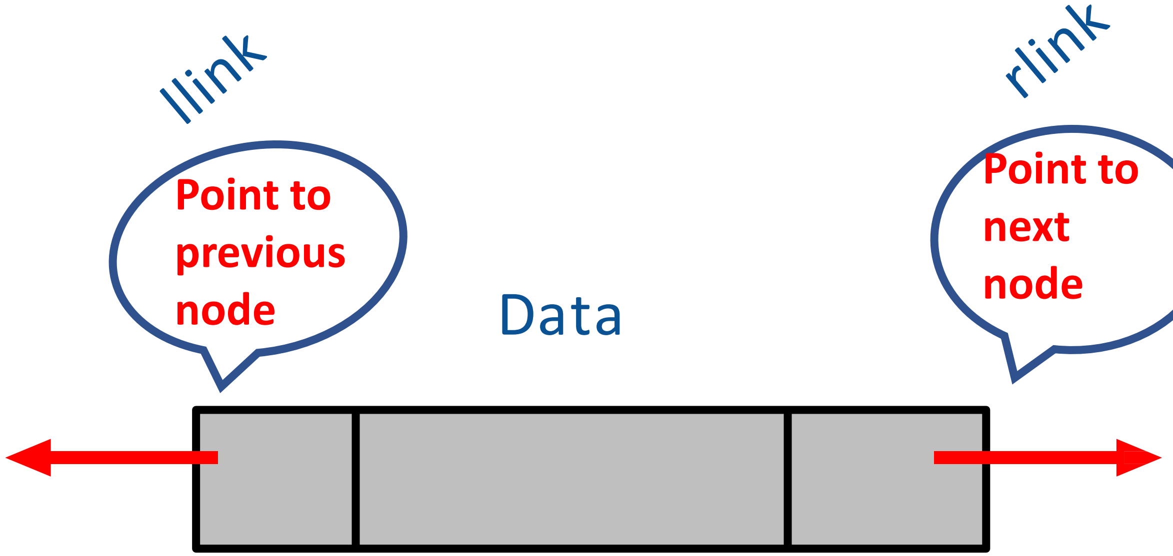


Doubly Linked List Vs Singly Linked List

- Advantages:
  - Can be traversed in either direction (may be essential for some programs)
  - Some operations, such as deletion and inserting before a node, become easier
- Disadvantages:
  - Requires more space
  - List manipulations are slower (because more links must be changed)
  - Greater chance of having bugs (because more links must be manipulated)

Doubly Linked List Node definition

```
struct node
{
  int data;
  node*llink;
  node*rlink;
};
```



Creating a node

- Allocate memory for the node dynamically

- If the memory is allocated successfully
  set the data part to user defined value
  set the llink (address of previous node) and rlink (address of next node) part to NULL

NULL	20	NULL
llink	data	rlink

Inserting a node

There are 3 cases

- Insertion at the beginning
- Insertion at the end
- Insertion at a given position

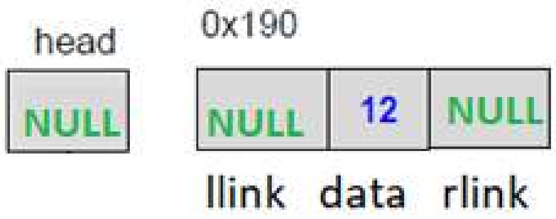
Insertion at the beginning

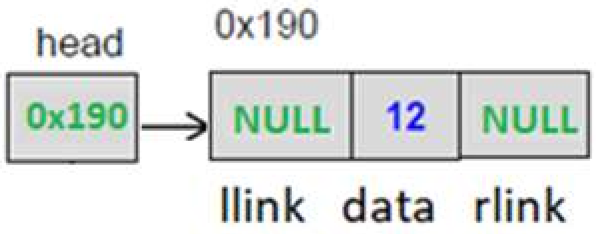
What all will change

If the linked list empty(case 1)
Head/Start pointer
else (case2)

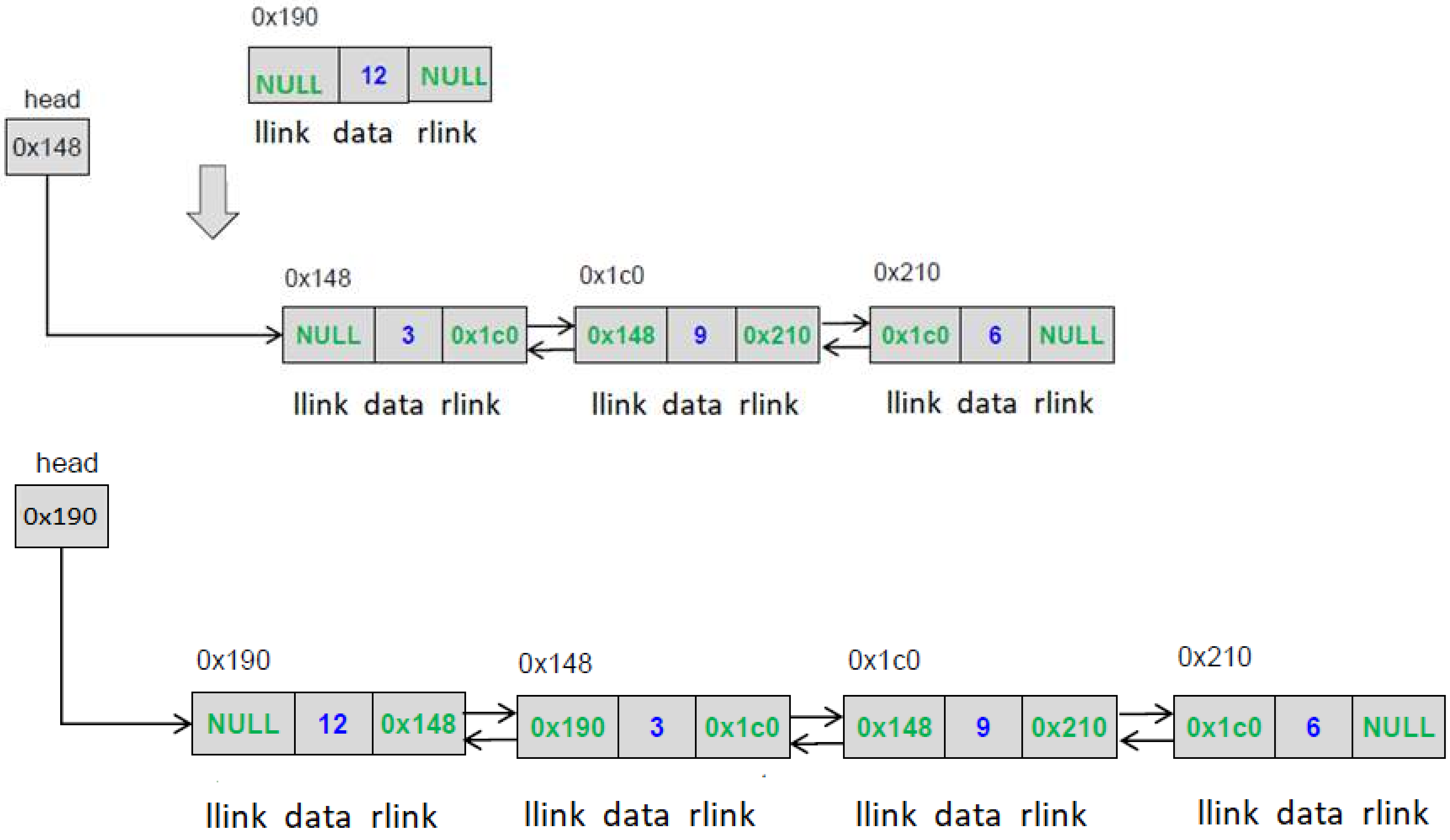
- Head/Start pointer
- New front's llink and rlink
- Old front's llink

Insertion at the beginning (Case1)





Insertion at the beginning(Case 2)



PES UNIVERSITY ONLINE

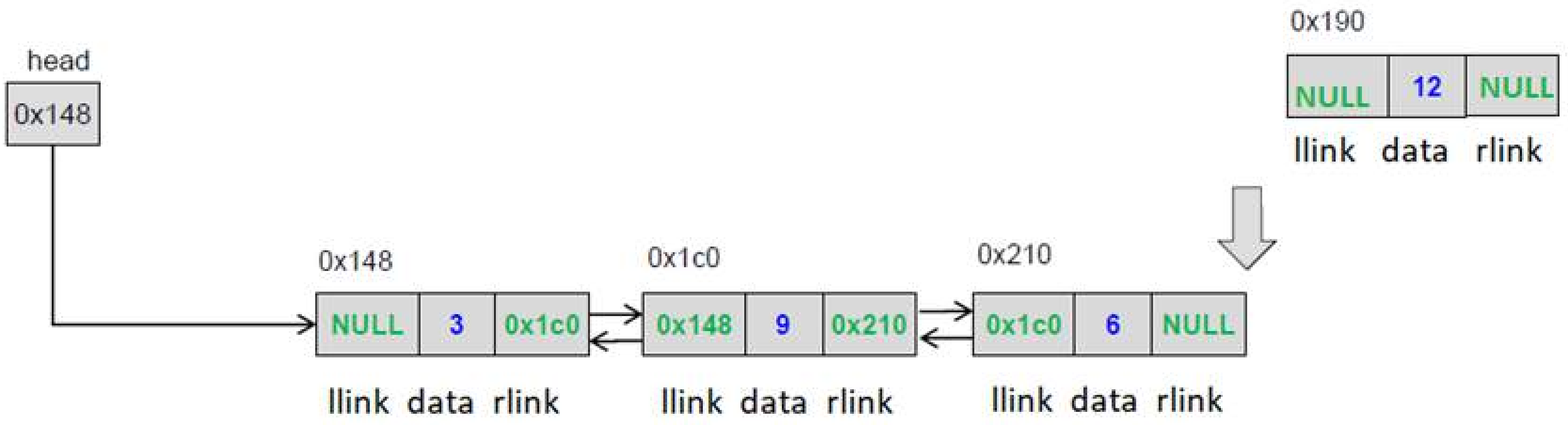
Insertion at the end

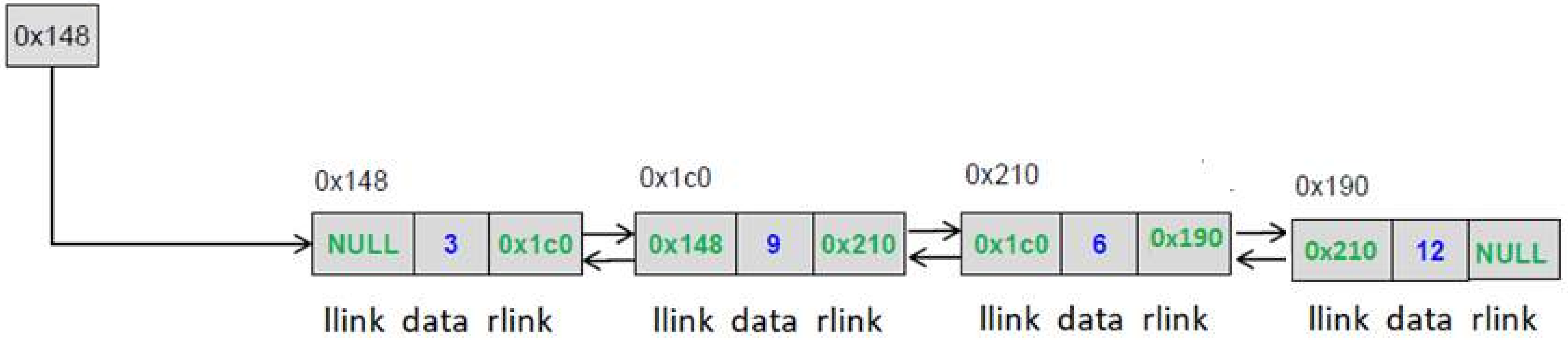
What all will change

If the linked list empty(same as case 1 of insert at front)
Head/Start pointer(case 2)
else

- Last node's rlink
- New node's llink

Insertion at the end





Insertion at the given position

- Create a node

If the list is empty

- make the start pointer point towards the new node;

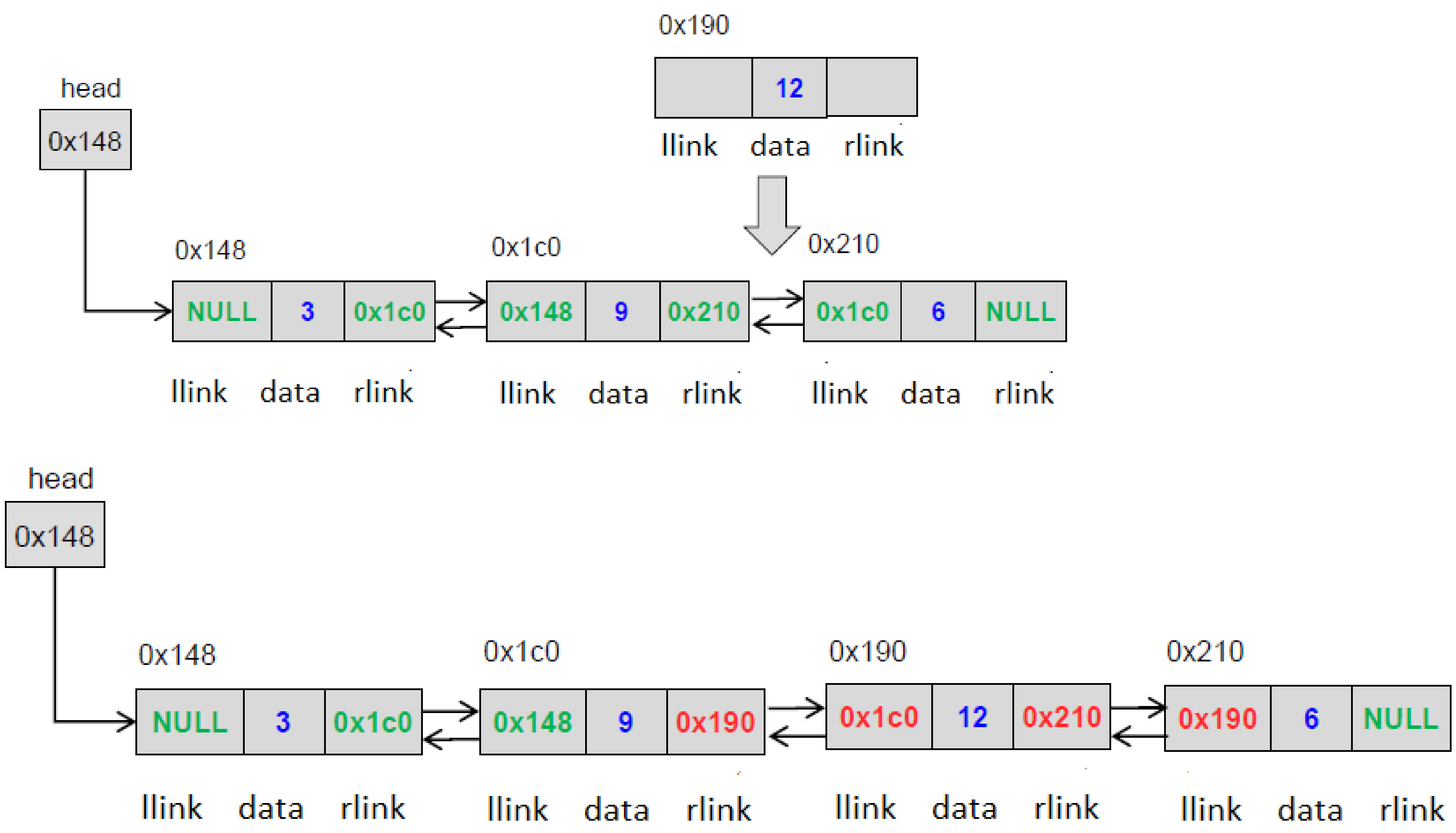
Else

- Traverse the linked list to reach given position
- Keep track of the previous node

If it is an intermediate position

- Change previous node rlink to point to the newnode
- Newnode's llink to point to previous node and rlink to point to the next node
- Next node llink to point to the newnode

Insertion at the given position



Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

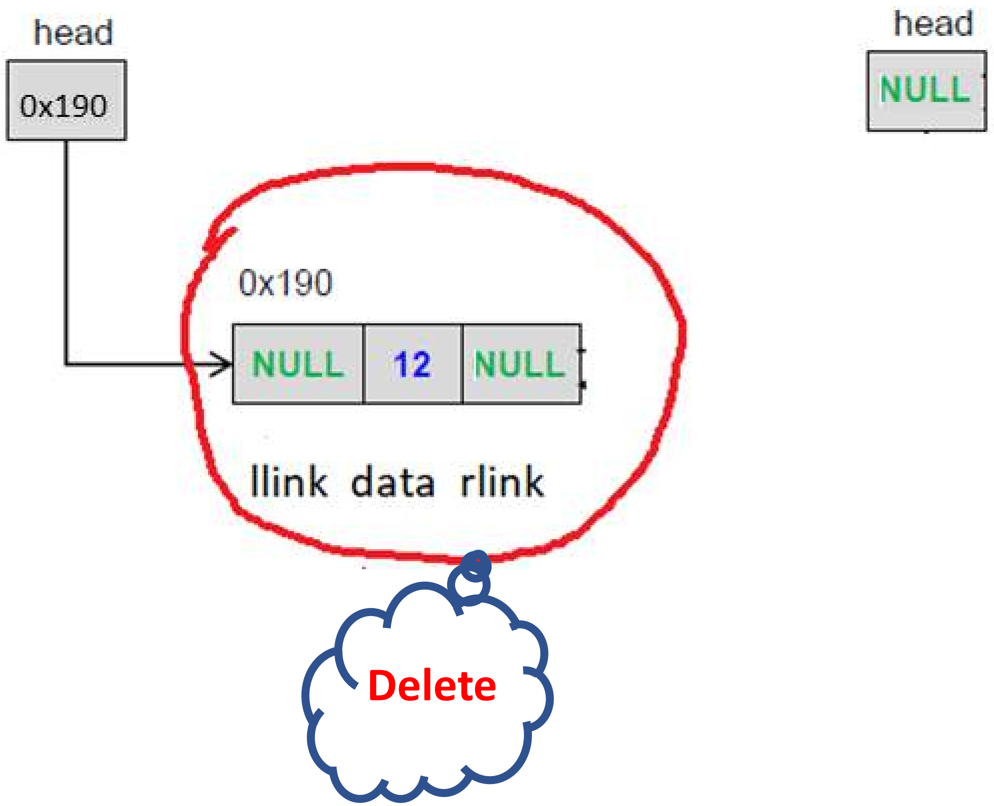
Deleting first node

What will change??

- Case I : Empty Linked List
- Case II : Linked list with a single node
  first node gets freed up
  head points to NULL
- Case III : Linked List with more than one node
  Second node llink gets changed to NULL
  first node gets freed off
  head points to second node

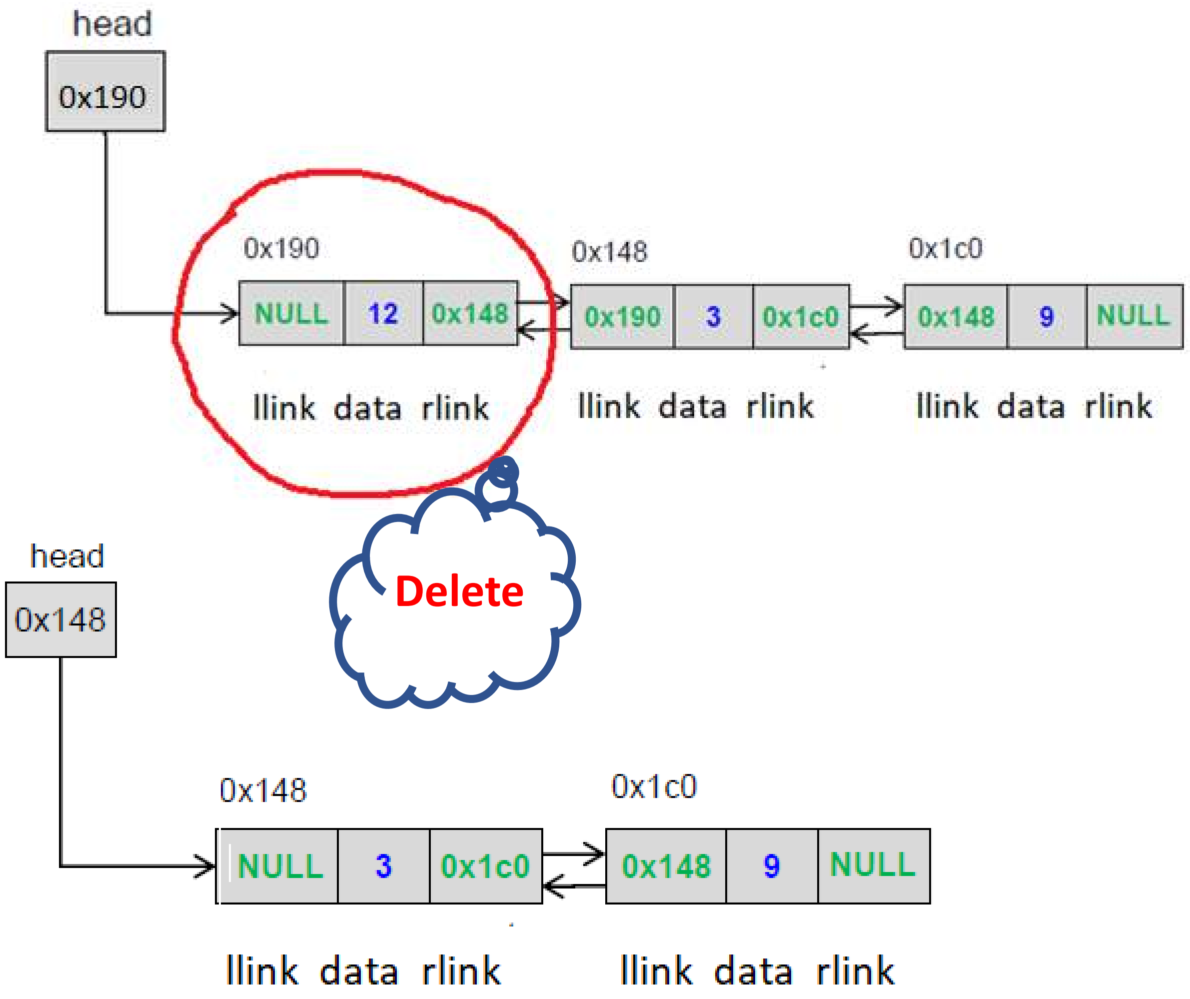
Deleting first node

- Case II : Linked list with a single node



Deleting first node

- Case III : Linked List with more than one node



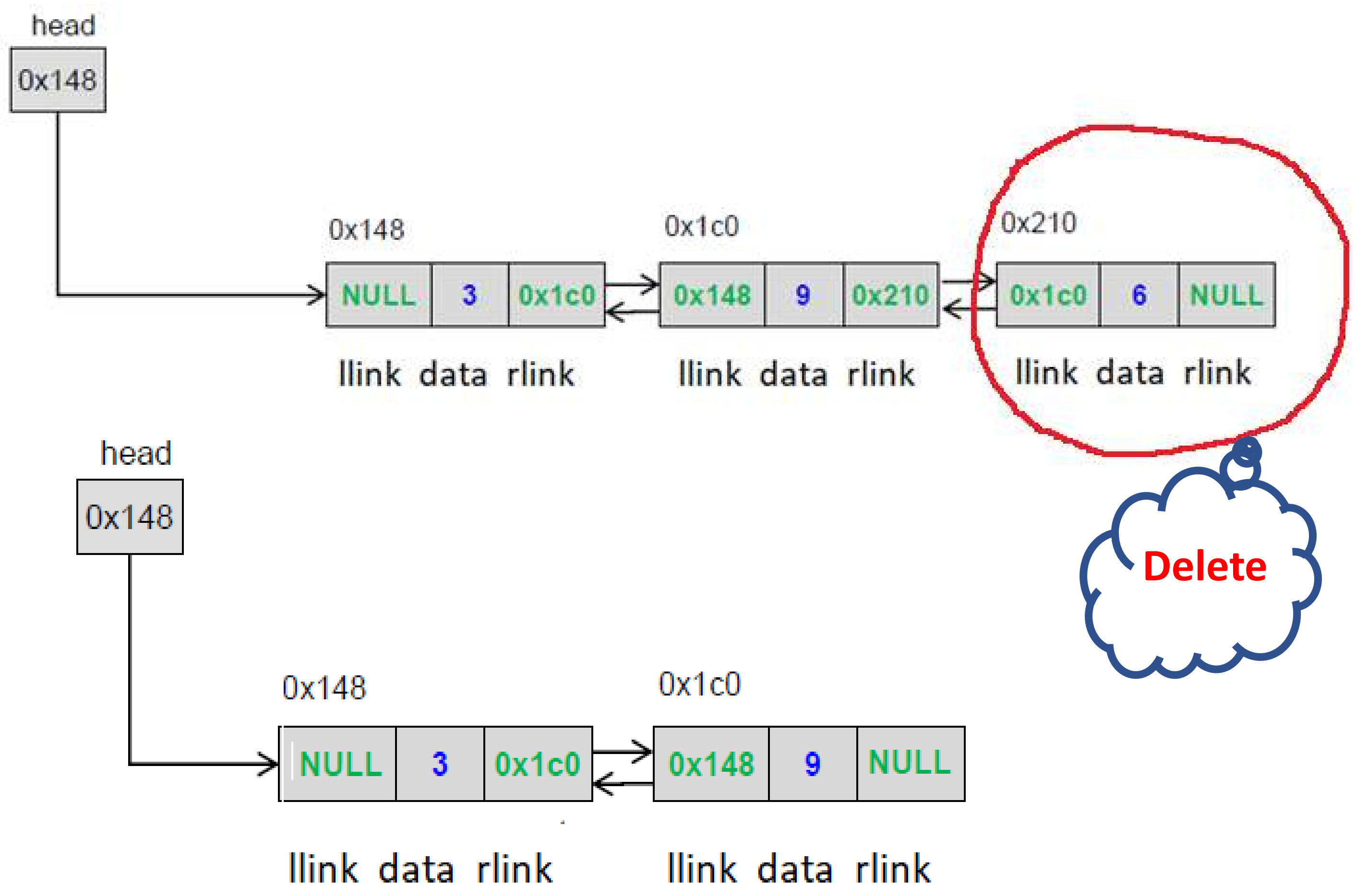
Deleting last node

What will change??

- Case I : Empty Linked List
- Case II : Linked list with a single node
  first node gets freed up
  head points to NULL
- Case III : Linked List with more than one node
  Second last node rlink point to NULL
  last node gets freed up

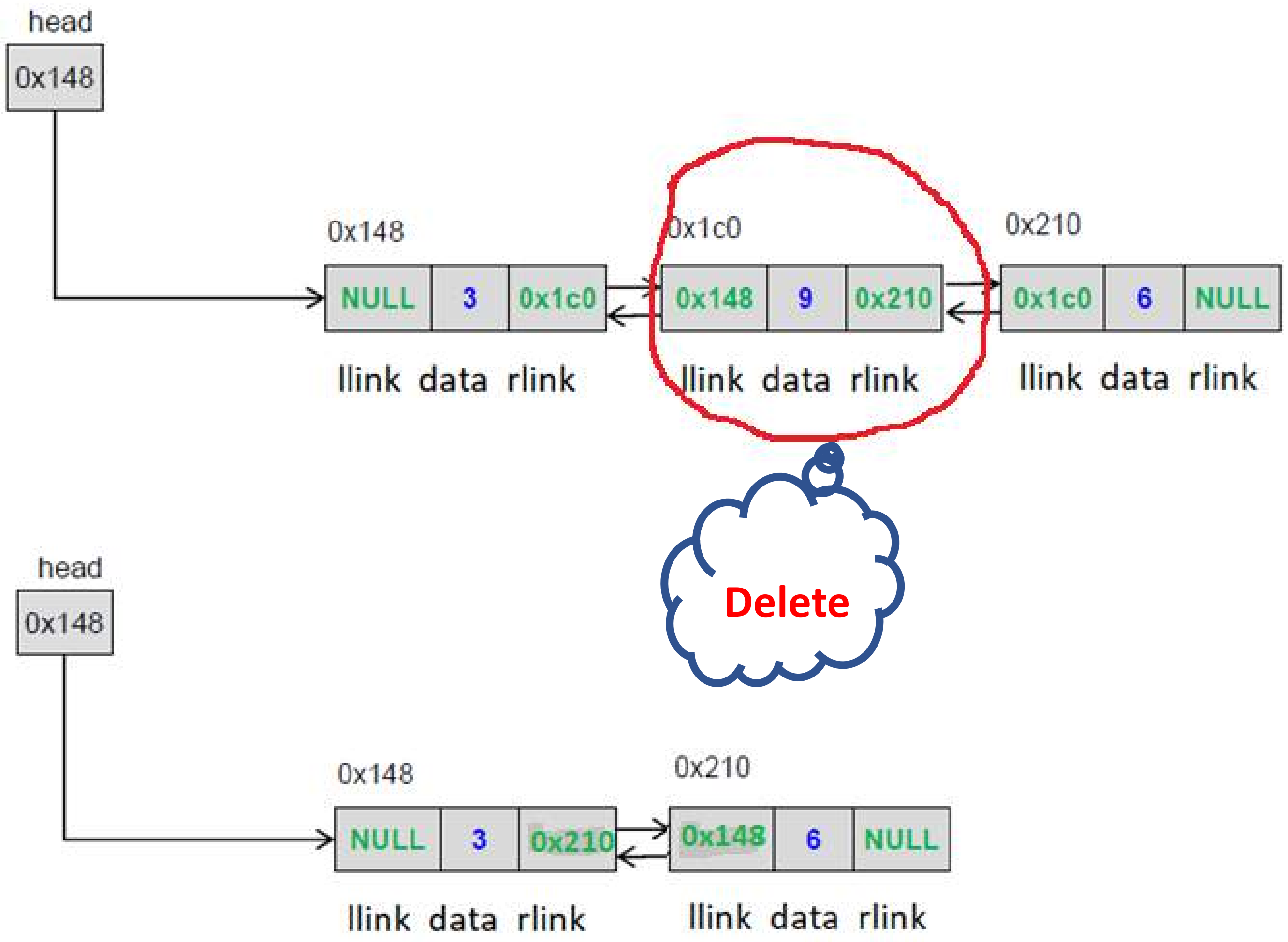
Deleting last node

- Case II : Linked List with more than one node



Deleting a node at intermediate position

- Case II : Linked List with more than one node



Doubly Linked List insert operation

Apply the concepts to implement following operations for a Doubly linked list

- reverse a doubly linked list
- Find the node pairs with a given sum in a doubly linked list
- Insert a node after a node with a given value
- Remove duplicate nodes from a doubly linked list

Vandana M L
Assistant Professor, Department of Computer Science

vandanamd@pes.edu

DATA STRUCTURES AND ITS APPLICATIONS

Vandana M L

Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Doubly Linked List

Vandana M L
Department of Computer Science and Engineering

Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

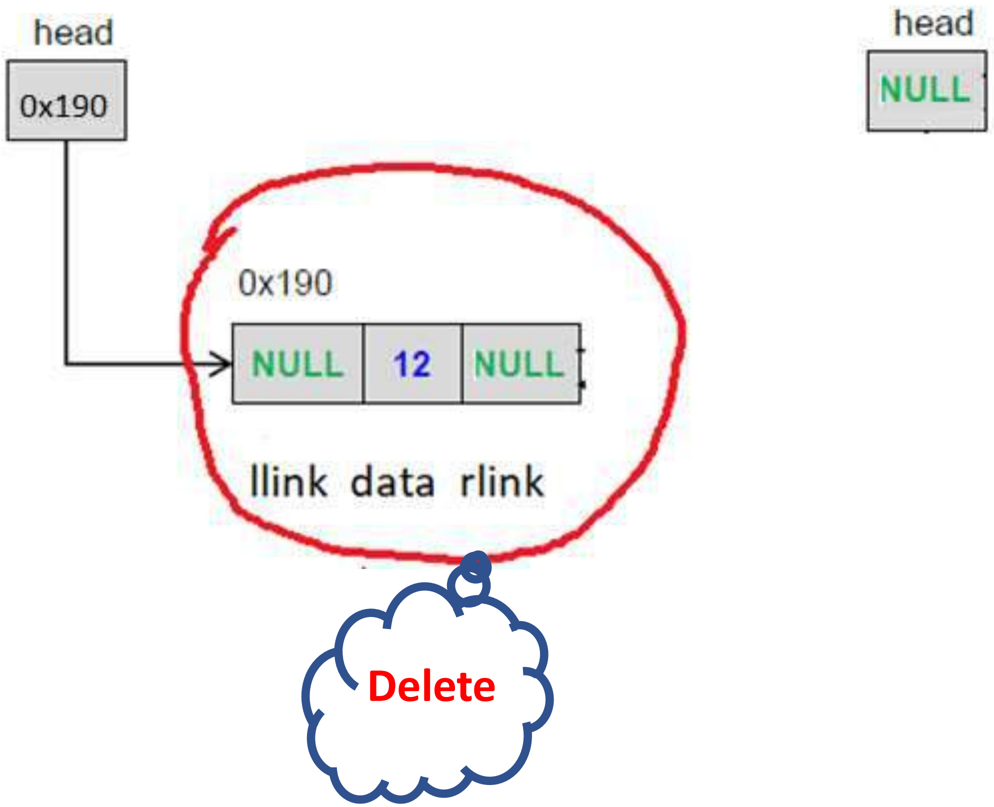
Deleting first node

What will change??

- Case I : Empty Linked List
- Case II : Linked list with a single node
  first node gets freed up
  head points to NULL
- Case III : Linked List with more than one node
  Second node llink gets changed to NULL
  first node gets freed off

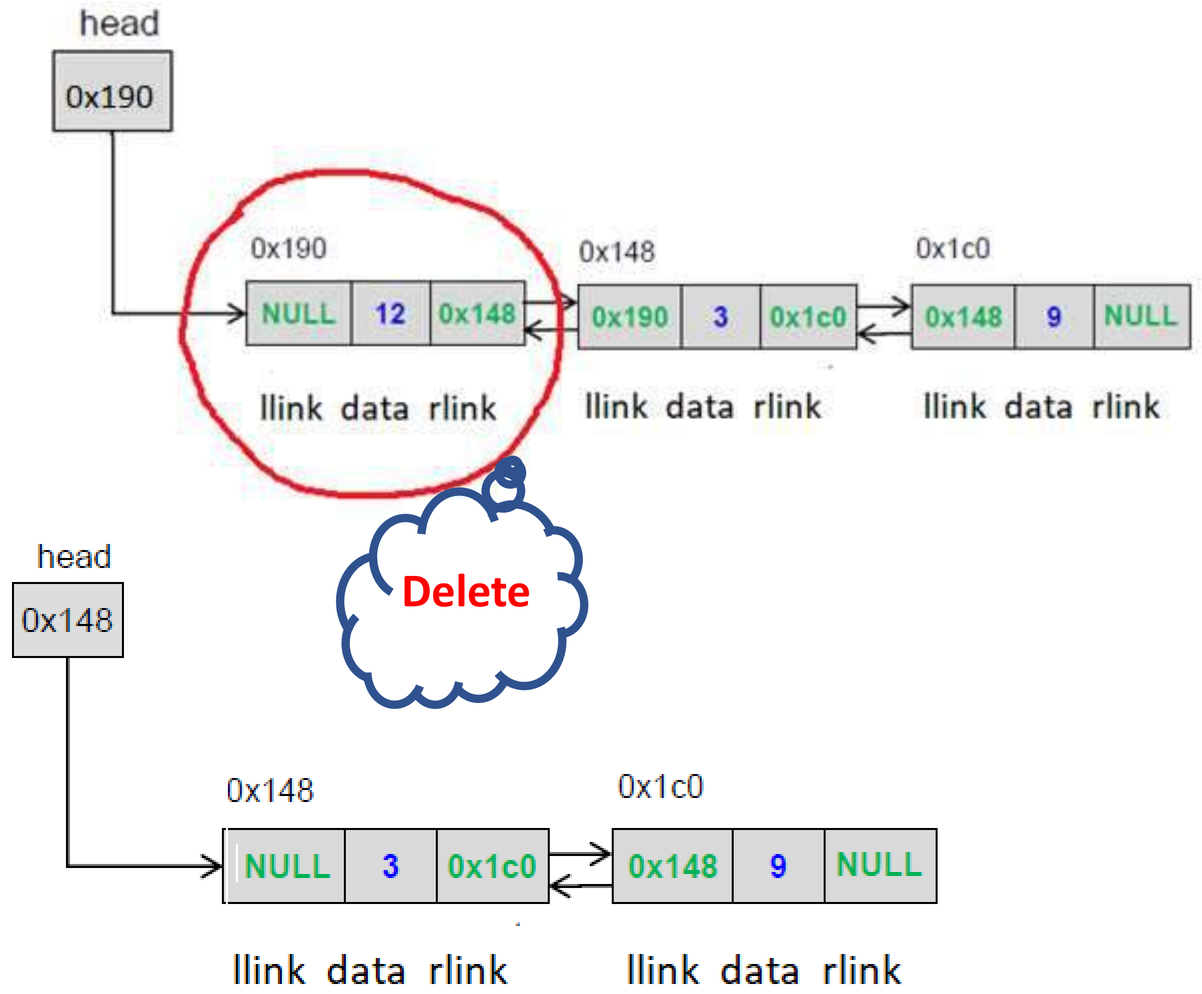
Deleting first node

- Case II : Linked list with a single node



Deleting first node

- Case III : Linked List with more than one node



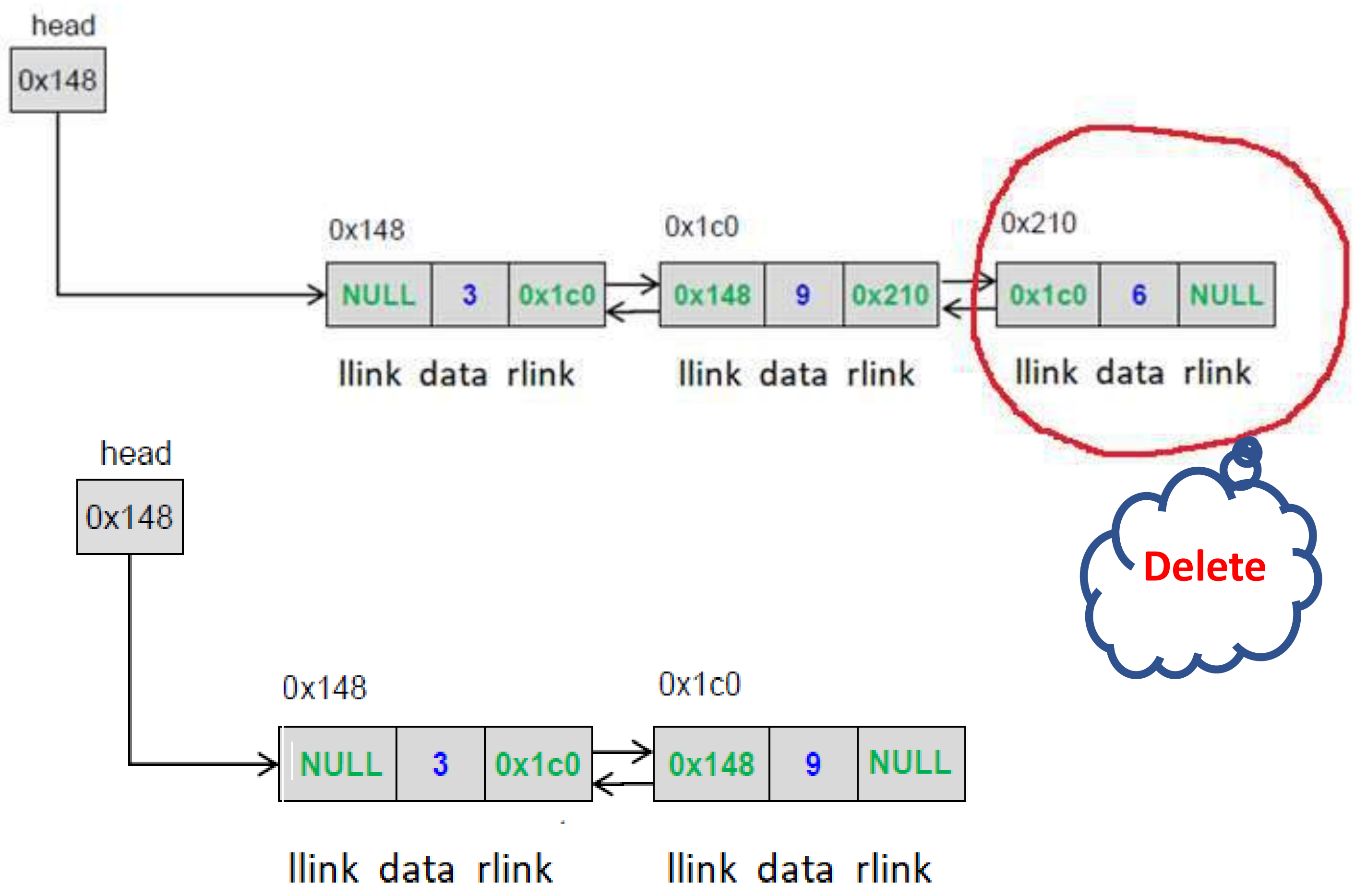
Deleting last node

What will change??

- Case I : Empty Linked List
- Case II : Linked list with a single node
  first node gets freed up
  head points to NULL
- Case III : Linked List with more than one node
  Second last node rlink point to NULL
  last node gets freed up

Deleting last node

- Case II : Linked List with more than one node



Deleting a node at intermediate position

- Traverse list to find the desired position, keep track of the previous node

If position is found

If position is 1

- Delete from front

else

If it is last position

- Delete from end

else

if intermediate position

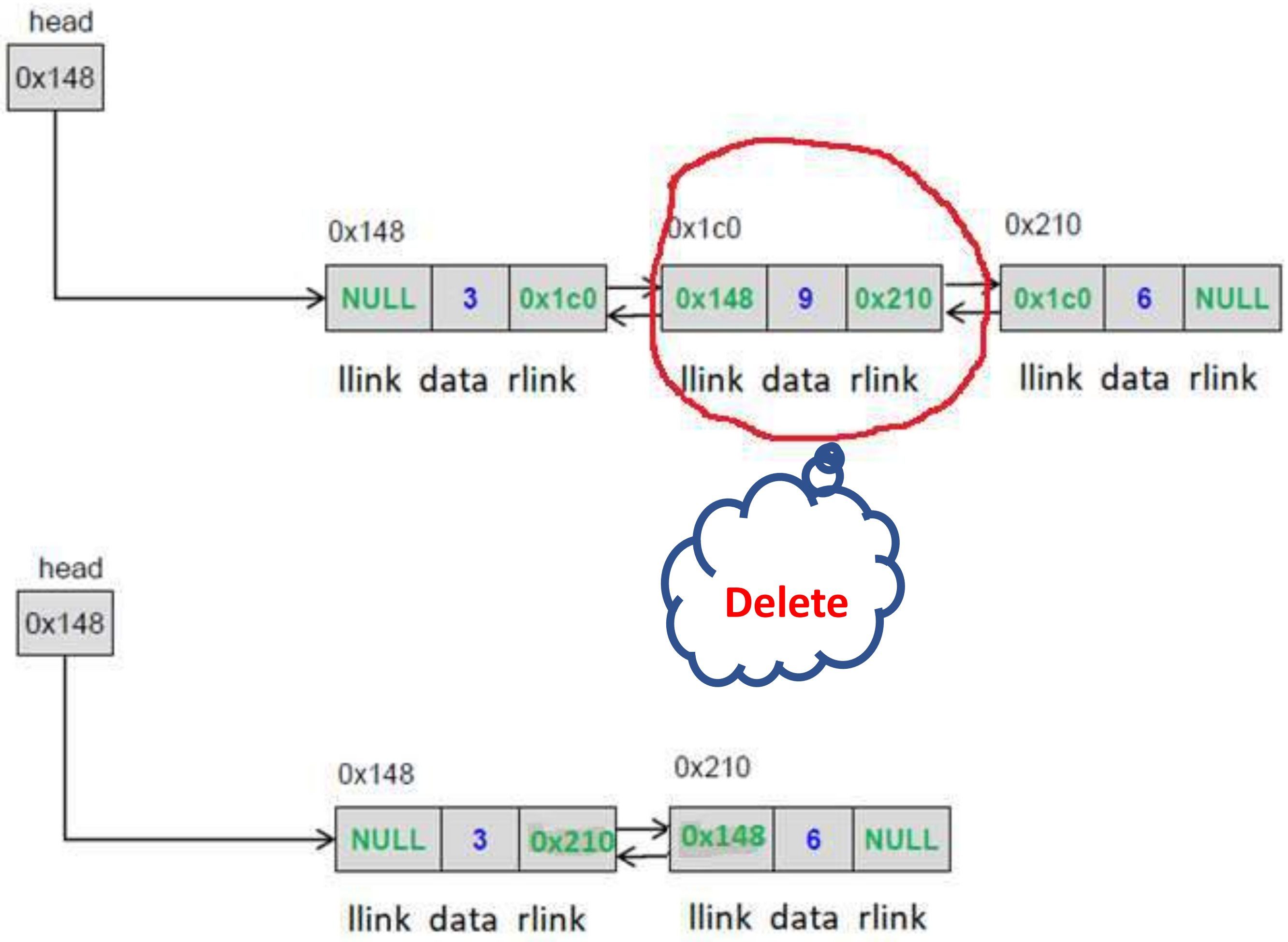
- Change previous node rlink to rlink of current node
- Change llink of node following current node to previous node
- Delete current node

else

invalid position

Deleting a node at intermediate position

- Case II : Linked List with more than one node



Doubly Linked List insert operation

Apply the concepts to implement following operations for a Doubly linked list

- reverse a doubly linked list
- Remove duplicate nodes from a doubly linked list
- Delete a node with a given key value from doubly linked list

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

+91 7411716615

DATA STRUCTURES AND ITS APPLICATIONS

Vandana M L

Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List

Vandana M L

Department of Computer Science and Engineering

Circular linked list is a linked list where all nodes are connected to form a circle.

- Circular Singly Linked List

- Circular Doubly Linked List

  With additional head node
  Without additional head node

Circular Linked List Operations

Insert a node

- Insert at front
- Insert at end
- Insert at a position
- Ordered insertion

Delete node

- Delete front node
- Delete end node
- Delete a node from position
- Delete a node with a given value

Additional

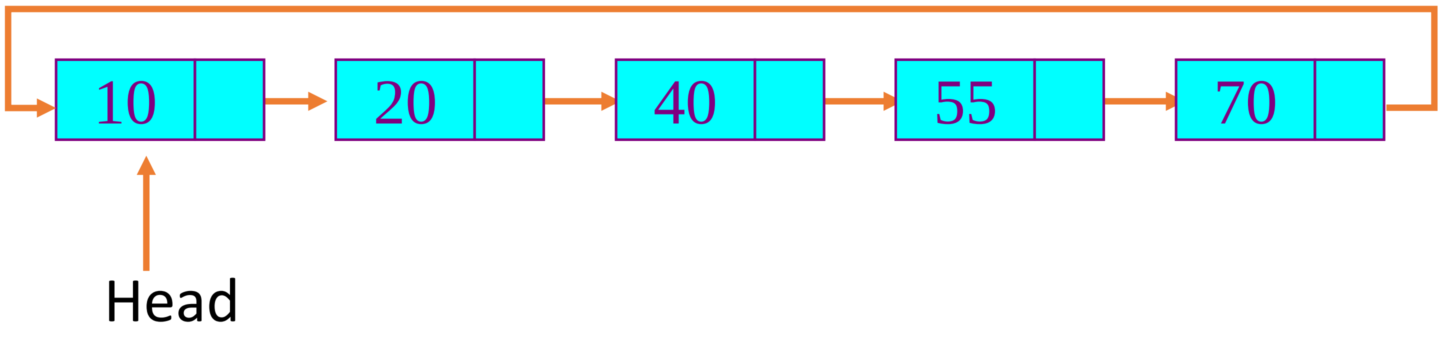
- Display list
- Concatenate two list
- reverse a list

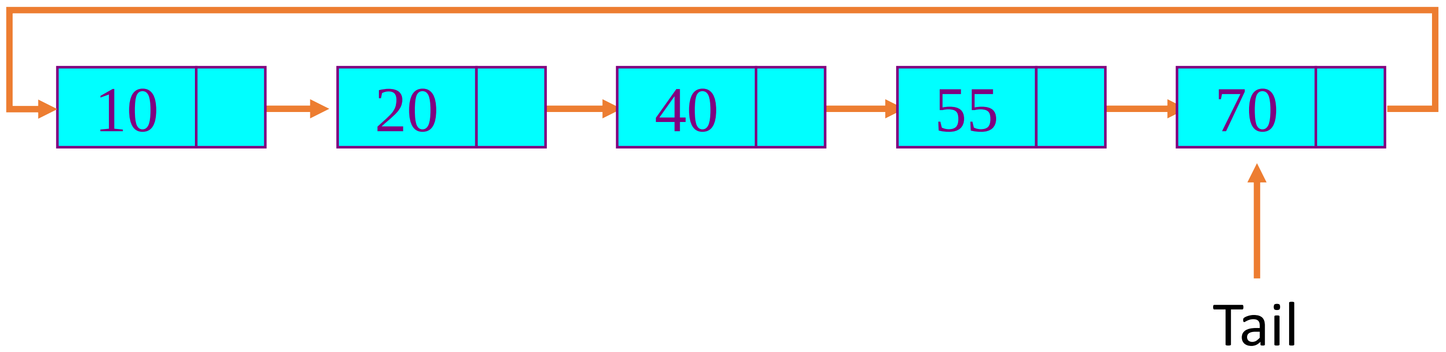
Circular Linked List: Applications

- Useful for implementation of queue, eliminates the need to maintain two pointers as in case of queue implementation using arrays
- Circular linked lists are useful for applications to repeatedly go around the list like playing video and sound files in “looping” mode
- Advanced data structures like Fibonacci Heap Implementation
- Plays a key role in linked implementation of graphs

Circular Singly Linked List

- It supports traversing from the end of the list to the beginning by making the last node point back to the head of the list
- A Tail pointer is often used instead of a Head pointer





Circular Singly Linked List Node Definition

```
#include <iostream>
using namespace std;

struct Node{
  int data;
  struct Node* next;
};
typedef struct node csll_node;
```

PES UNIVERSITY ONLINE

Insertion at the beginning

Insert at the front of linked list

- Create a node

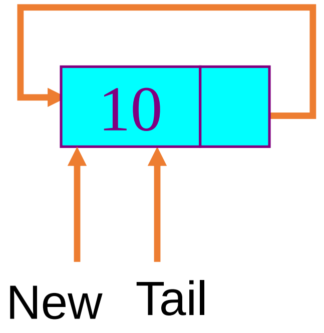
If the list is empty

- make the tail pointer point towards the new node

Else

- Change the new node link field to point to the first node
- Change the last node link to point to the new node

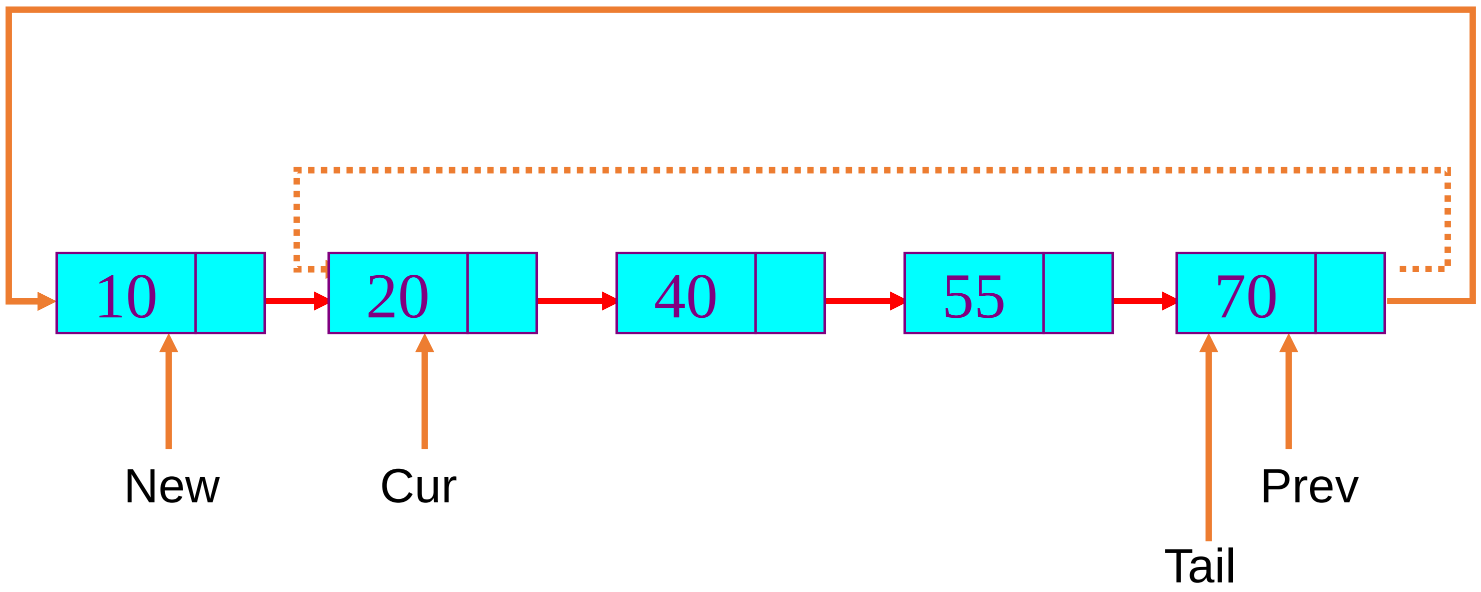
Insertion into an empty list



Insert to head of a Circular Linked List

New->next = Cur; → New->next = Tail->next;

Prev->next = New; → Tail->next = New;

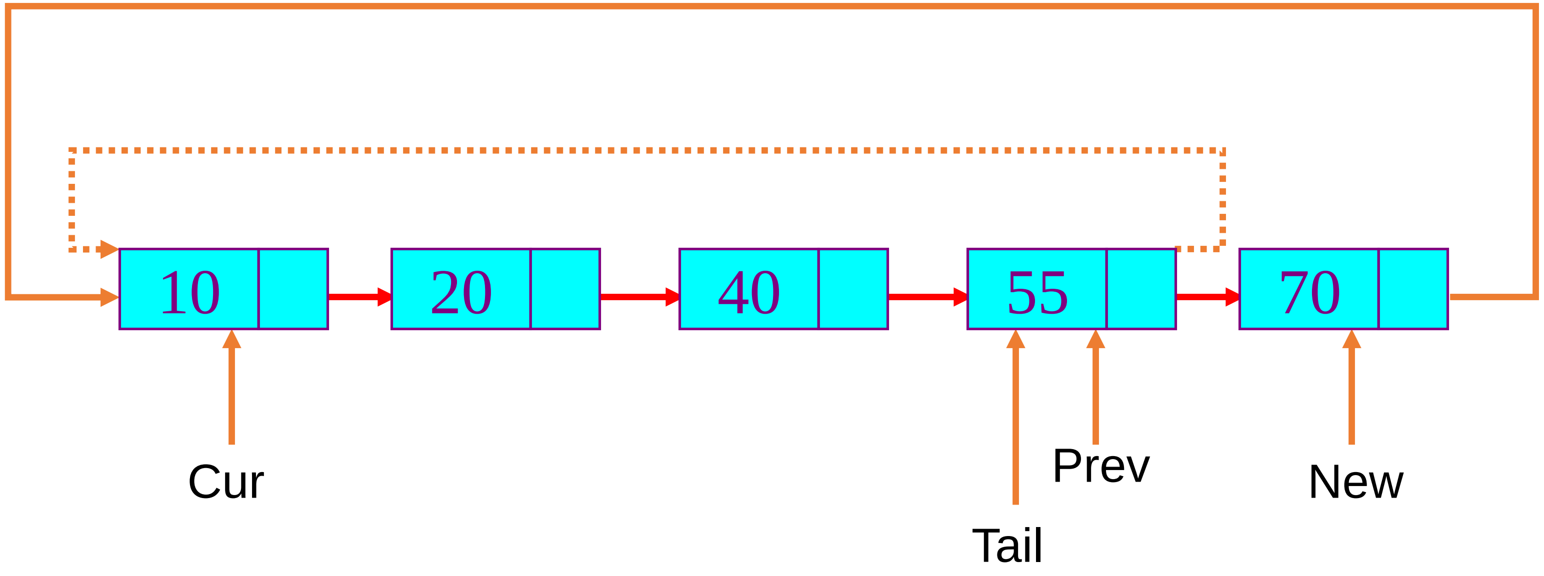


Insert to the end of a Circular Linked List

New->next = Cur; → New->next = Tail->next;

Prev->next = New; → Tail->next = New;

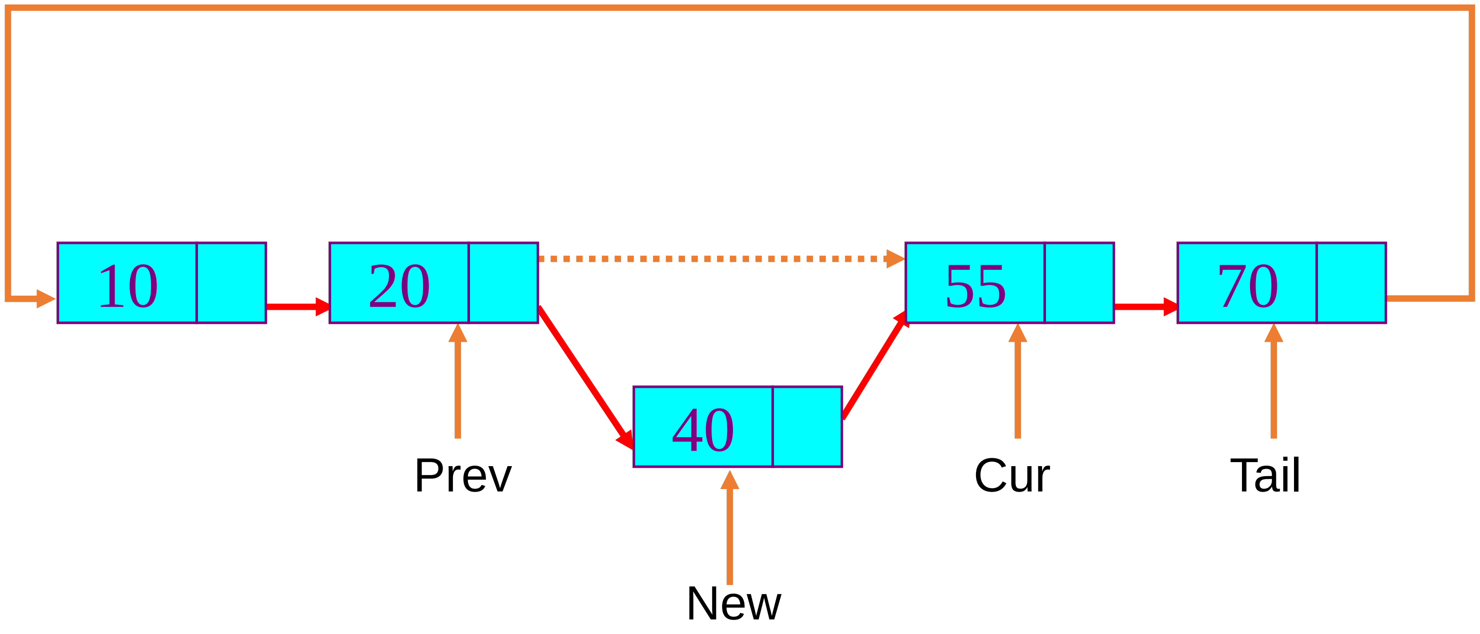
Tail = New;



Insert to the middle of Circular Linked List

New->next = Cur;

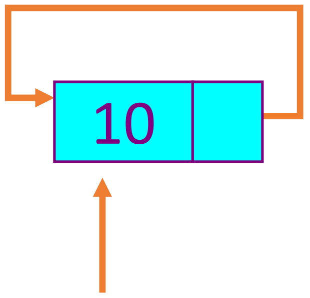
Prev->next = New;



Delete a node from a single-node Circular Linked List

Tail = NULL;

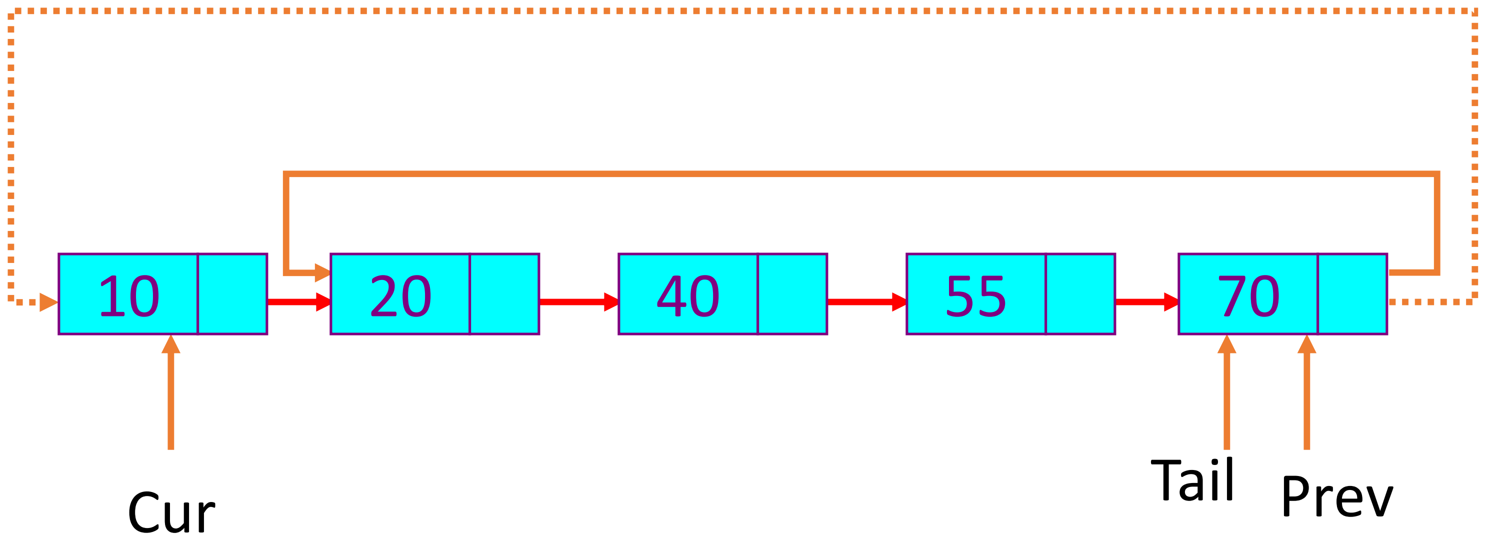
free(Cur);



Tail = Cur = Prev

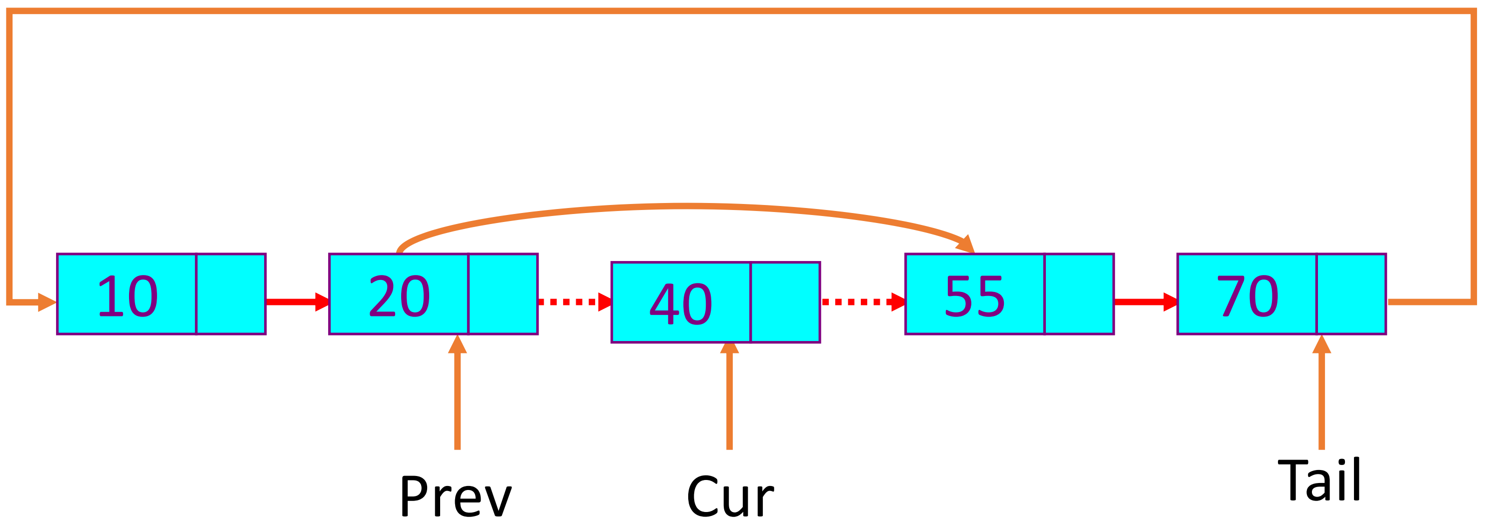
Delete the head node from a Circular Linked List

```
Prev->next = Cur->next;        // same as: Tail->next = Cur->next

free(cur);
```



Delete a middle node Cur from a Circular Linked List

```
Prev->next = Cur->next;

Free(Cur);
```



Circular Singly Linked List operations

Apply the concepts to implement following operations for a singly linked list

- insert a node after a given node(pointer)
- Insert a node after a node with a given value

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

+91 7411716615

Data Structures and itsApplications

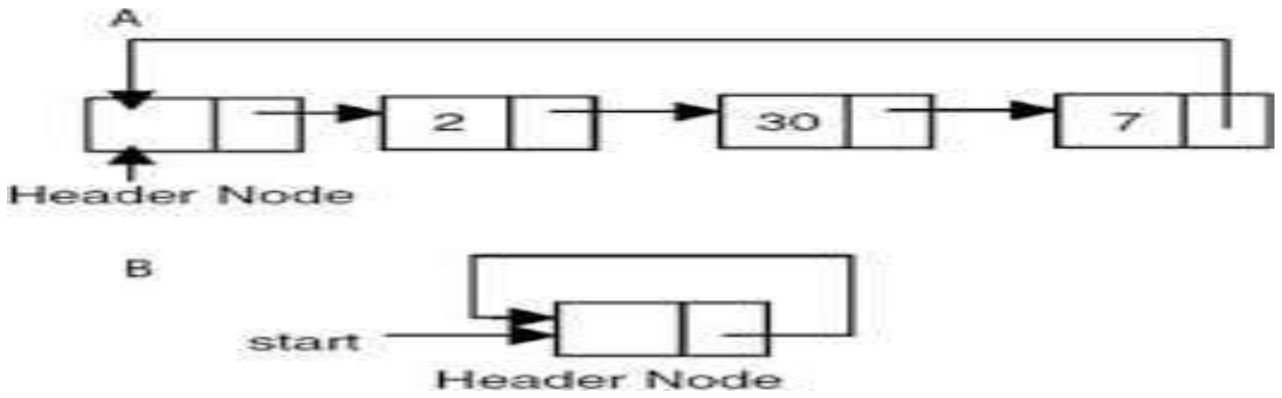
Dinesh Singh
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Circular Linked Lists

Dinesh Singh

Department of Computer Science & Engineering



- The first node in the list is the header node.
- The address part of the last node points to the header node
- Circular list does not have a natural first or the last node
- An External pointer points to the header node and the one following this node is the first node.

Implementation of some operations on Circular linked Lists with header node

- Insert at the head of a list
- Insert at the end of the List
- Delete a Node given its value

Note: head is a pointer to the header node and the following node is the first node

Creating Header node

```
struct node *create_head()
 {
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   temp->data=0; // keeps the count of nodes in the list
   temp->next=temp;
   return temp;
  }
```

Algorithm to insert a node at the head of the list
insert_head(p,x)
//p pointer to header node, x element to be inserted
//x gets inserted after the header node

allocate memory for the node
initialise the node
//insert the new node after the header node
Copy the value of the next part of the header node into the next part of the new node
Copy the address of the new node into next part of the header node

Operations on Circular Linked Lists with header node

```
void insert_head(struct node *p,int x)
//p points to the header node, x element to be inserted
 {
  struct node *temp;
  //create node and initialise
   temp=(struct node*)malloc(sizeof(struct node));
   temp->data=x;
// next part of new node points to the node after the header node
   temp->next=p->next;
   p->next=temp; //next part of header node points to the new node
   p->data++;
  }
```

Operations on Circular Linked Lists with header node

Algorithm to insert a node at the end of the list

insert_tail(p,x)
//p pointer to header node, x element to be inserted
allocate memory for the node
initialise the node

move to the last node
//insert the new node after the last node
Copy the address of the header node into next of new node
Copy the address of the new node into the next of last node

Algorithm to insert a node at the end of the list

```
void insert_tail(struct node *p,int x)
 {
  struct node *temp,*q;
   temp=(struct node*)malloc(sizeof(struct node));
   temp->data=x;

   q=p->next; // go the first node

   while(q->next!=p) // move to the last node
    q=q->next;

 temp->next=p;// copy address of header node into next of new node
 q->next=temp; // copy the address of new node into next of the last node
 p->data++; // increment the count of nodes in the list
 }
```

Algorithm to delete a node given its value

delete_node(p,x)

//p pointer to header node, x element to be deleted

move forward until the node to be deleted is found

or header node is reaches

If(node found)

  delete the node by adjusting the pointers

else

  node not found // if header node is reached

Operations on Circular Linked Lists with header node

```
void delete_node(struct node *p, int x)
 {
  //p points to the header node, x is element to be inserted
   struct node *prev,*q;

   q=p->next; // go to the first node
   prev=p;
  //move forward until the data is found or header node is reached
   while((q!=p)&&(q->data!=x))
   {
    prev=q; // keep track of the previous node
    q=q->next;
   }
```

Operations on Circular Linked Lists with header node

```
if(q==p) // header node reached
    printf("Node not found..\n");
  else
  {
   prev->next=q->next; //delete the node
   free(q);
   p->data--; // decrement the count of nodes in the list
  }
}
```

THANK YOU

Dinesh Singh
Department of Computer Science & Engineering

dineshs@pes.edu

+91 8088654402

DATA STRUCTURES AND ITS APPLICATIONS

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Circular Doubly Linked List

Vandana M L
Department of Computer Science and Engineering

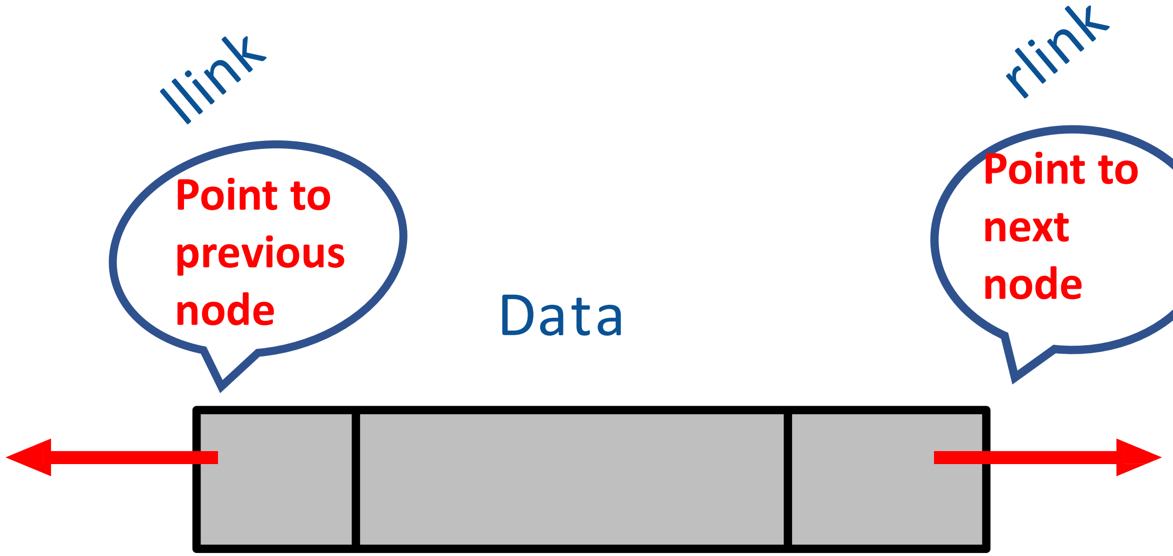
Node Structure Definition

A doubly linked list node contain three fields:

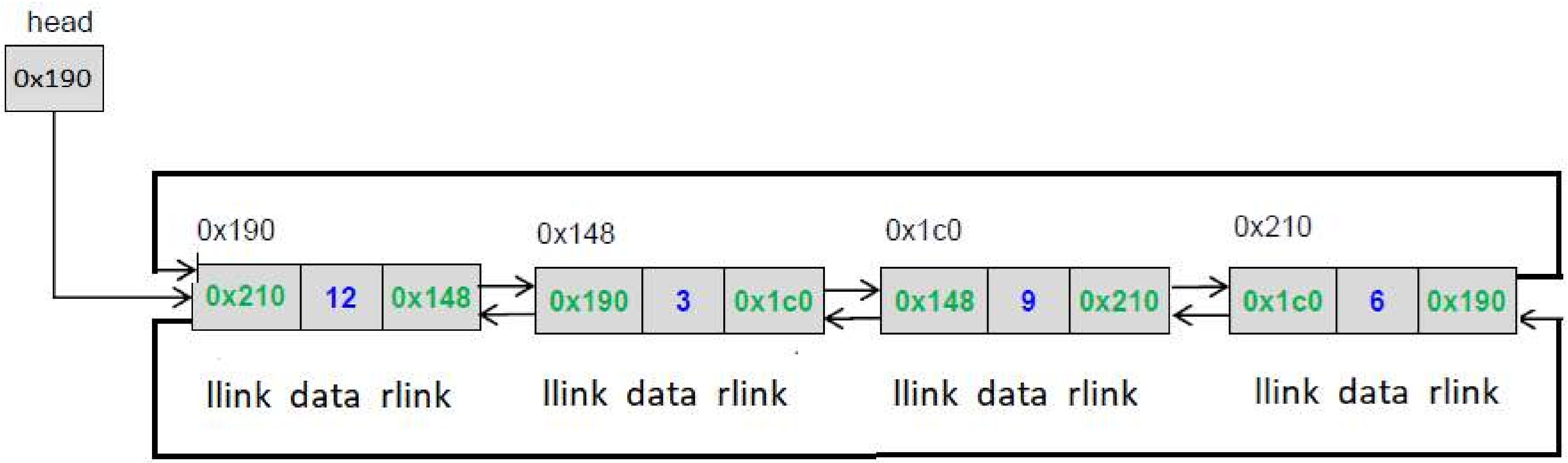
- Data
- link to the next node
- link to the previous node.

Node Structure Definition

```
struct node
{
   int data;
   struct node*llink;
   struct node*rlink;
};
```



Circular Doubly Linked List: Example



Creating a node

- Allocate memory for the node dynamically

- If the memory is allocated successfully
    - set the data part
    - set the llink and rlink to NULL

NULL	20	NULL

Inserting a node

There are 3 cases

- Insertion at the beginning
- Insertion at the end
- Insertion at a given position

Insertion at the beginning

What all will change

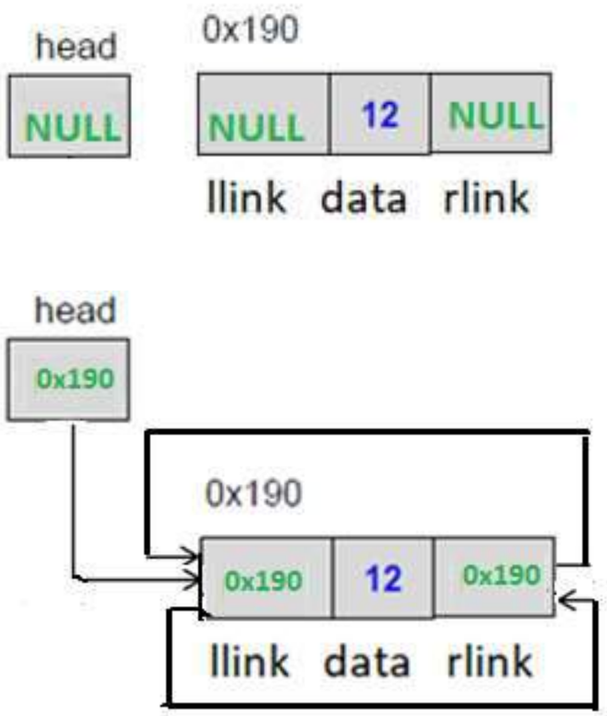
Case 1: linked list empty

- Head pointer

Case 2: linked list is not empty

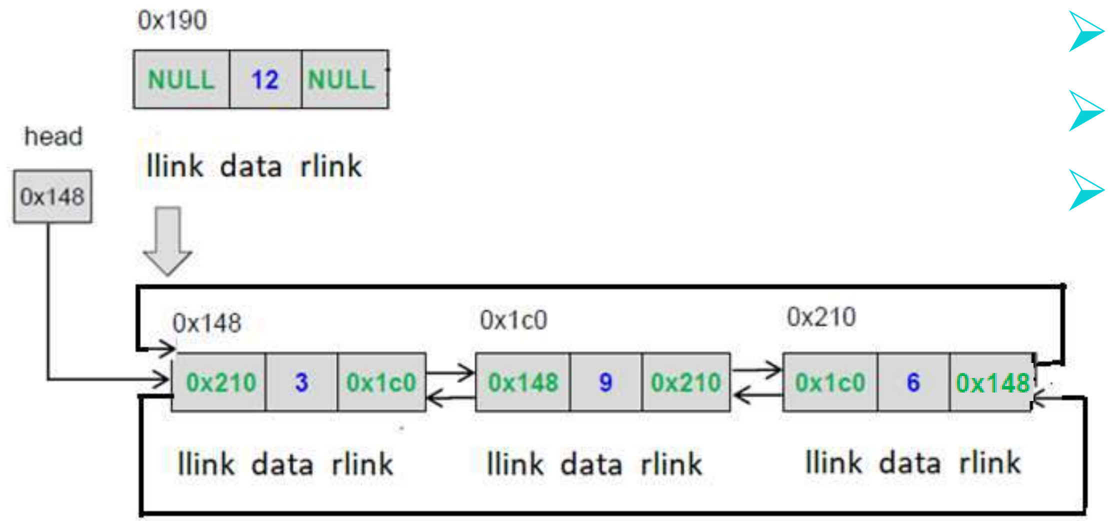
- Head pointer
- New front node's rlink and llink
- Old front node's llink
- Last node's rlink

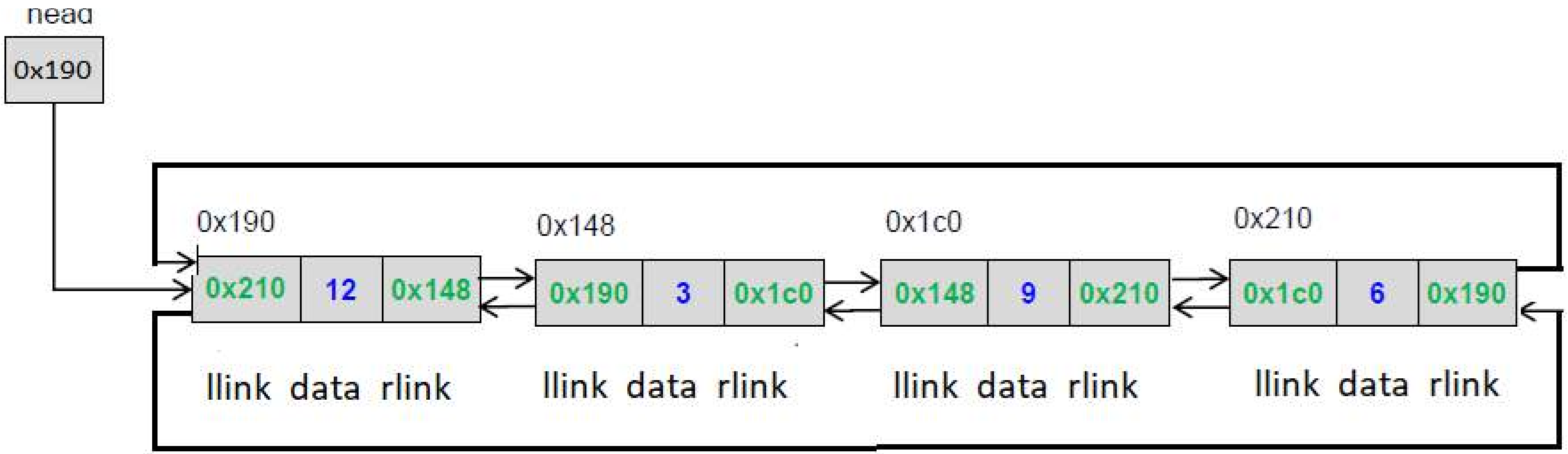
Insertion at the beginning (Case1)



Insertion at the beginning(Case 2)

- Head pointer
- New front node's rlink and llink
- Old front node's llink
- Last node's rlink





Insertion at the end

What all will change

Case 1: linked list empty

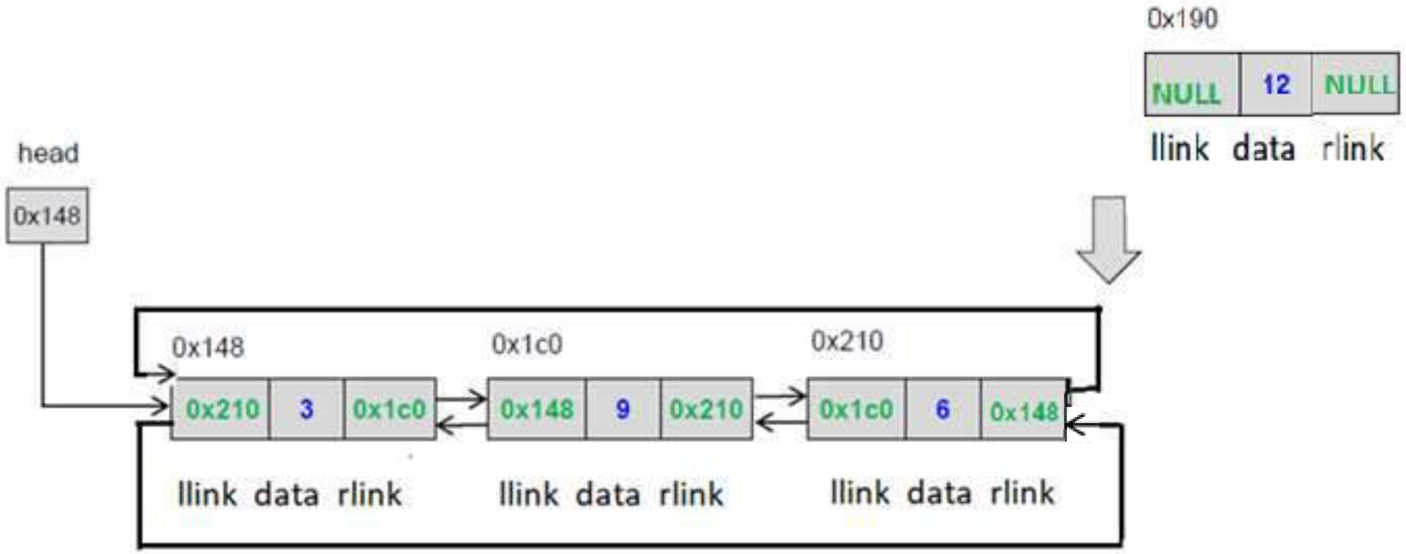
- Head pointer

Case 2: linked list is not empty else

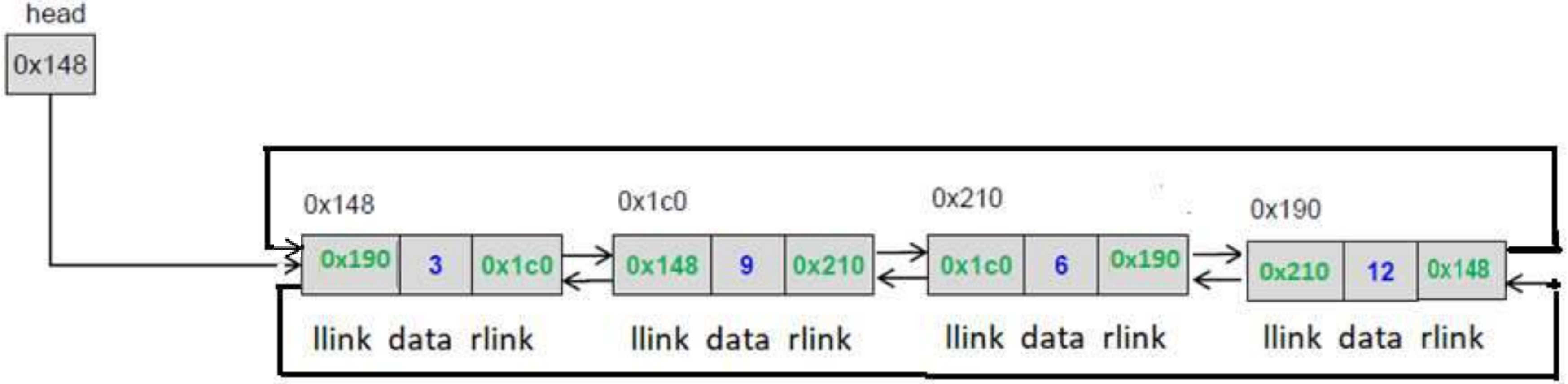
- Last node's rlink
- First node llink
- New node's llink and rlink

Circular Doubly Linked List Operations

Insertion at the end



node’s rlink
node llink
node’s llink and rlink



Insertion at the given position

- Create a node

If the list is empty

- make the start pointer point towards the new node;

Else

if it is first position

- Insert at front

else

- Traverse the linked list to reach given position
- Keep track of the previous node

If it is valid position

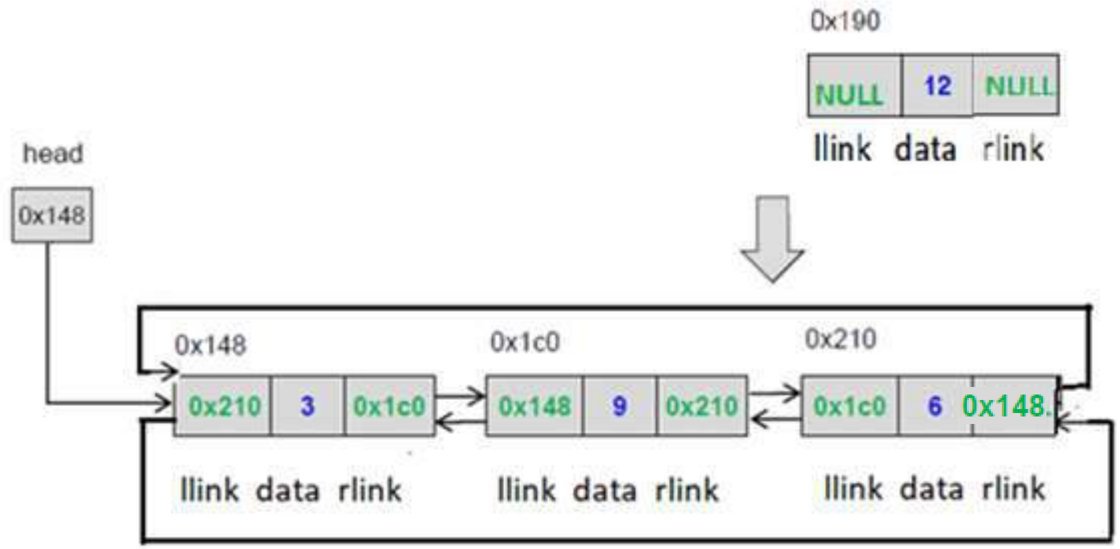
intermediate position

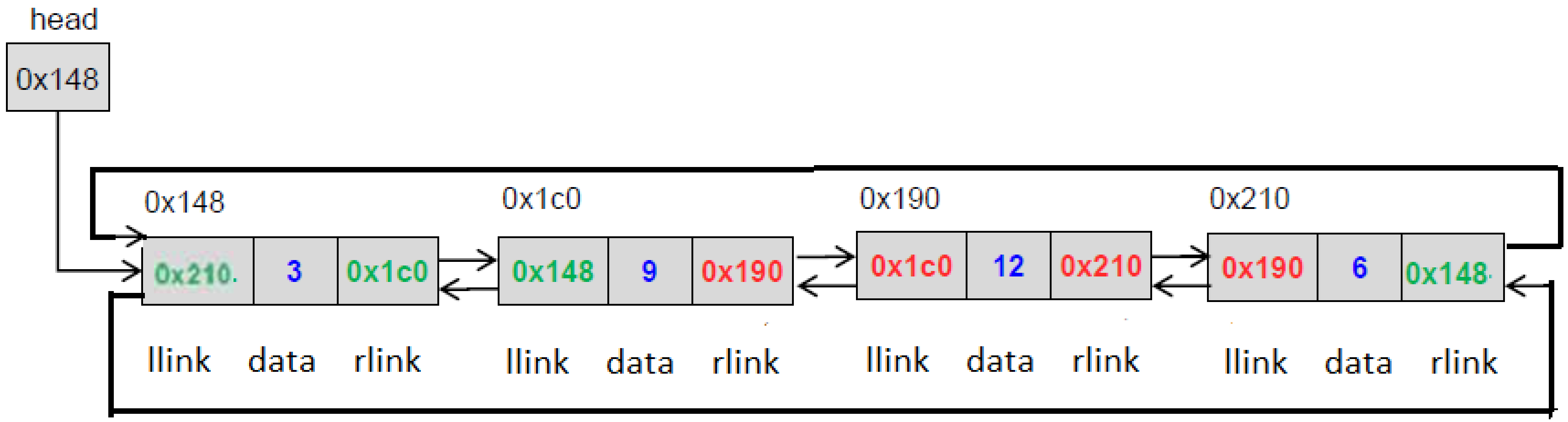
- Change link fields of current previous and intermediate node

last position

- insert at end

Insertion at the given position





Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

Deleting a node

There are 3 cases

- Deleting first node
- Deleting last node
- Deleting a node at a given position

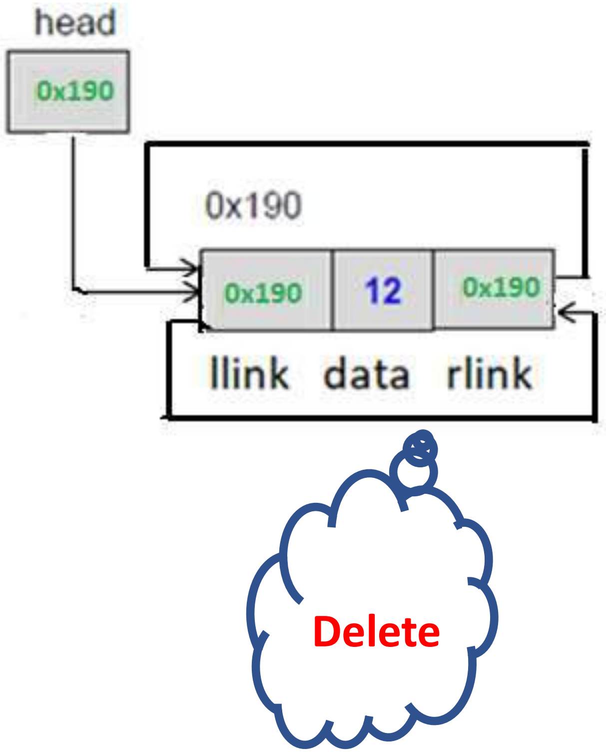
Deleting first node

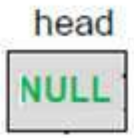
What will change??

- Case I : Empty Linked List
- Case II : Linked list with a single node
  first node gets freed up
  head points to NULL
- Case III : Linked List with more than one node
  Second node llink
  last node rlink
  first node gets freed off
  head pointer points to second node

Deleting first node

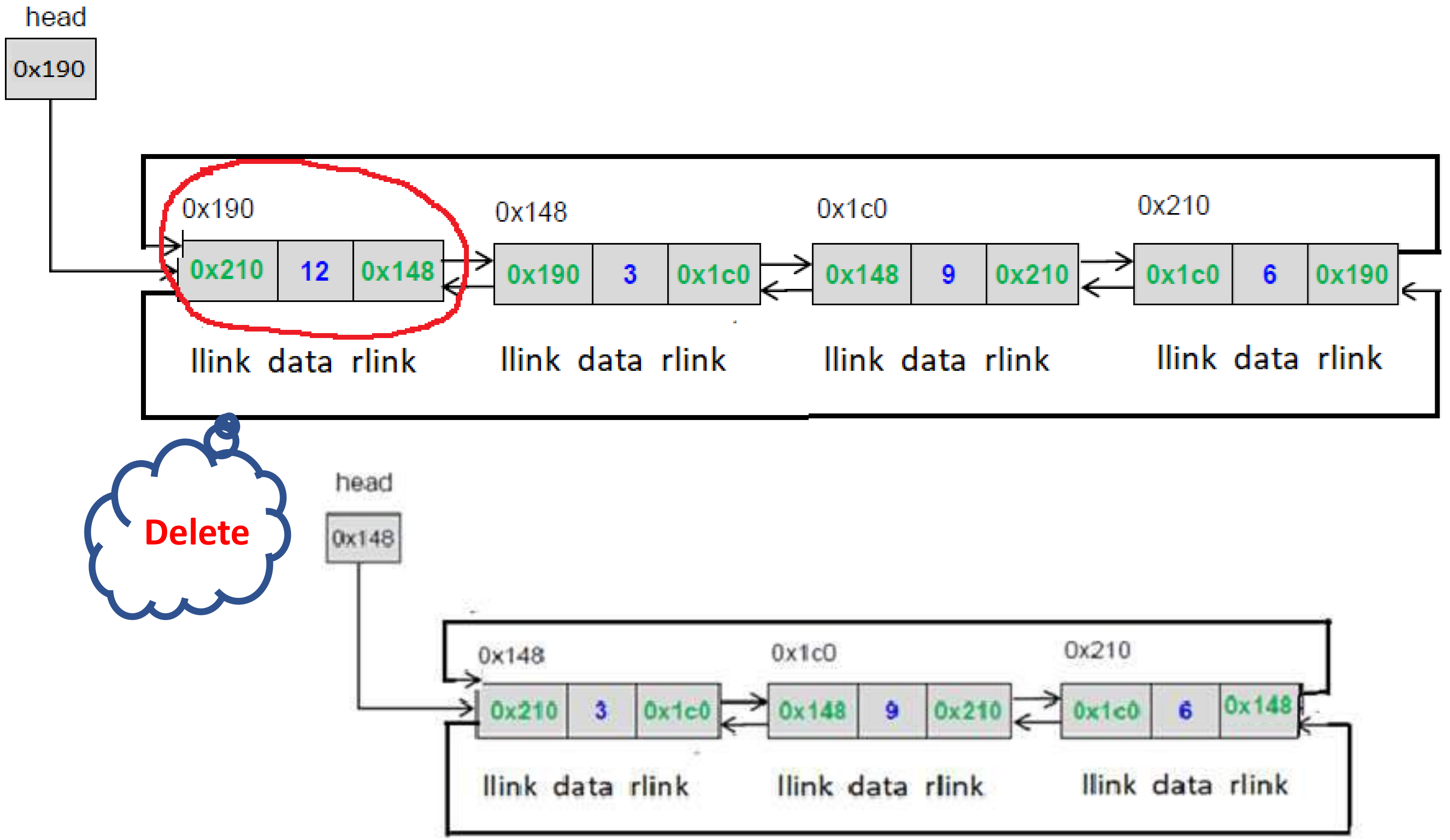
- Case II : Linked list with a single node





Deleting first node

- Case III : Linked List with more than one node



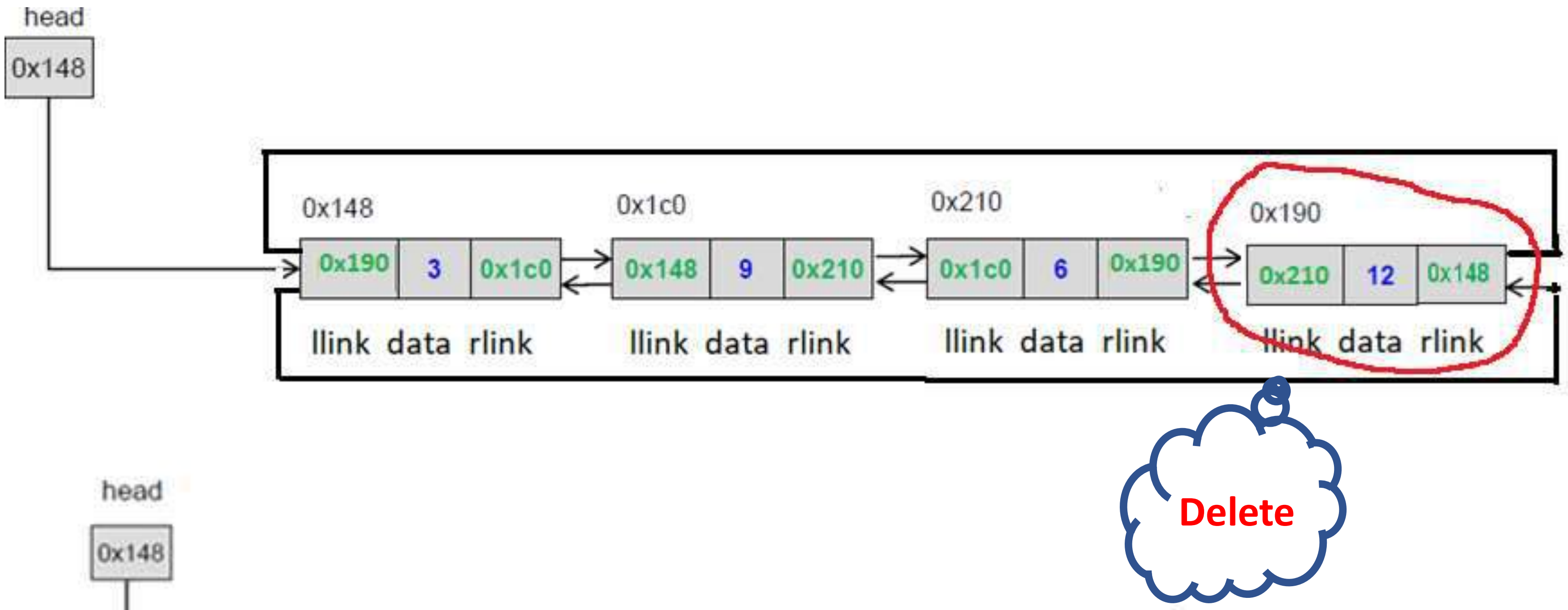
Deleting last node

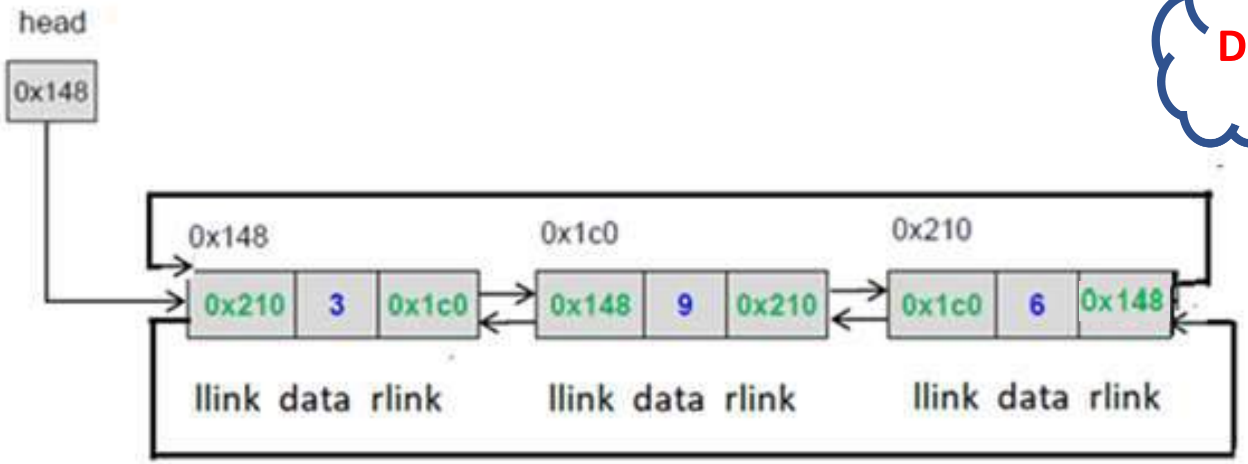
What will change??

- Case I : Empty Linked List
- Case II : Linked list with a single node
  - first node gets freed up
  - head points to NULL
- Case III : Linked List with more than one node
  - Second last node rlink
  - first node llink
  - last node gets freed up

Deleting last node

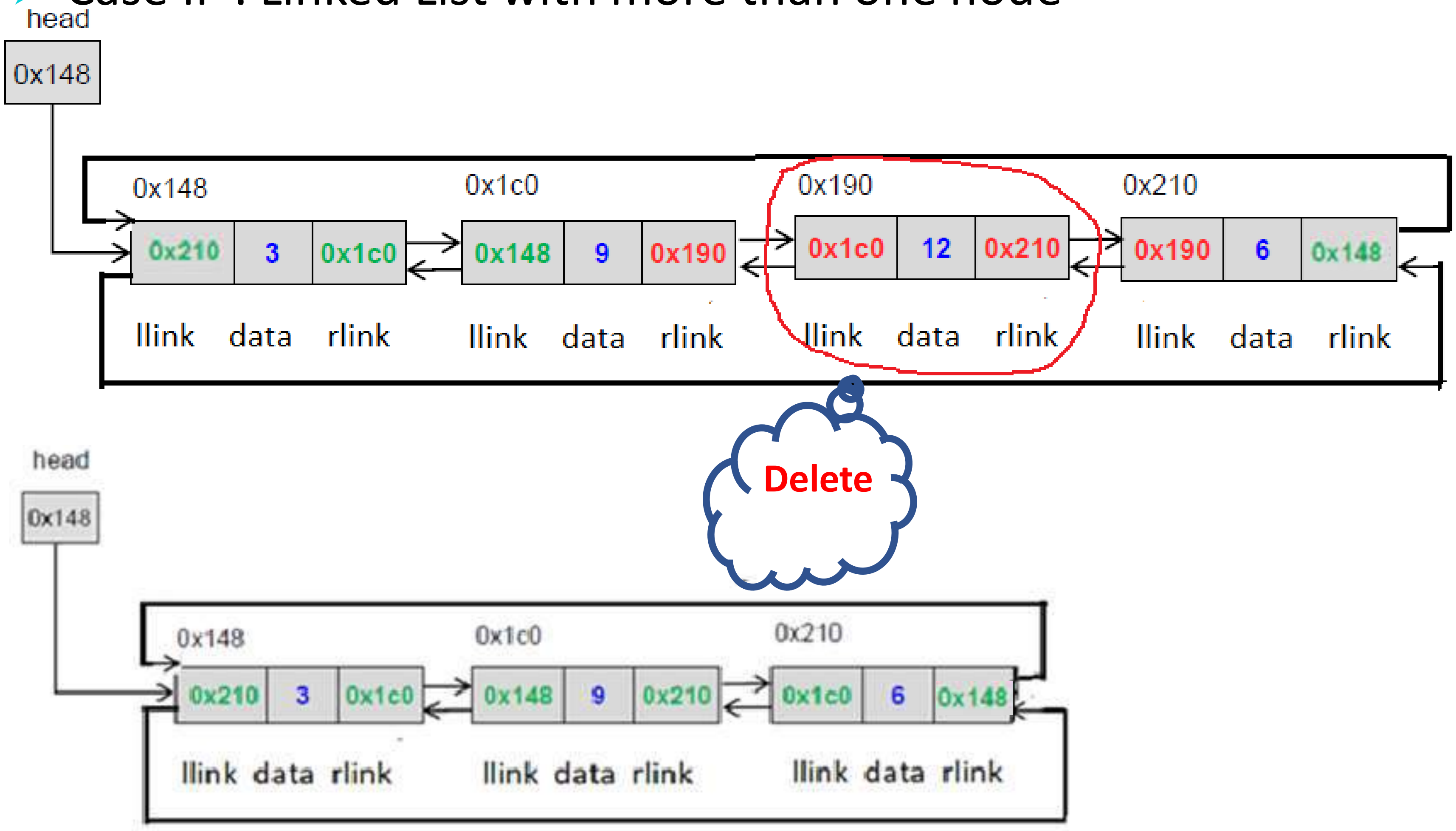
- Case II : Linked List with more than one node





Deleting a node at intermediate position

- Case II : Linked List with more than one node



Circular doubly Linked List operations

Apply the concepts to implement following operations for a doubly circular linked list

- Reverse list using recursion
- Search given element in the list
- Find the largest value in the list

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

+91 7411716615

DATA STRUCTURES AND ITS APPLICATIONS

Prof. Vandana M L

Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Multilist Representation

Prof. Vandana M L
Department of Computer Science and Engineering

Matrix ??

Two Dimensional data

1 1 3 0 4
1 3 5 1 0
9 0 5 1 0

Sparse Matrix??

More zero elements than non zero elements

0 0 3 0 0
0 0 5 1 0
0 0 0 0 0

Sparse Matrix Representation

- 2D Matrix

  results in lot of memory wastage as non zero elements are also stored
- Triple Notation

  Array representation
- Multilist Representation

  Linked representation hence size can be changed dynamically

Sparse Matrix Representation: Triple Notation

In triple notation sparse matrix is represented as an array of tuple values.
Each tuple consists of
<rowno columnno Value>

The first block in array block holds information regarding
<total no of rows, total no of columns ,value>

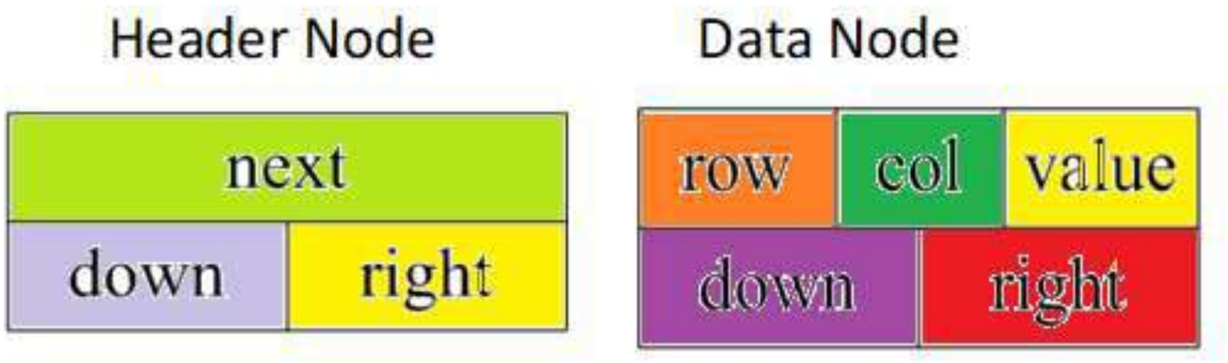
$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$

Triple Notation

Row No	Column No	Value
5	4	6
0	0	2
1	0	4
1	3	3
3	0	8
3	3	1
4	2	6

Node Structure

Two types of nodes are used



Node Structure Definition

```
#define MAX_SIZE 50 /* size of largest
matrix */
typedef enum {head, entry} tagfield;
typedef struct matrixNode * matrixPointer;
typedef struct entryNode {
int row;
int col;
int value; };
```

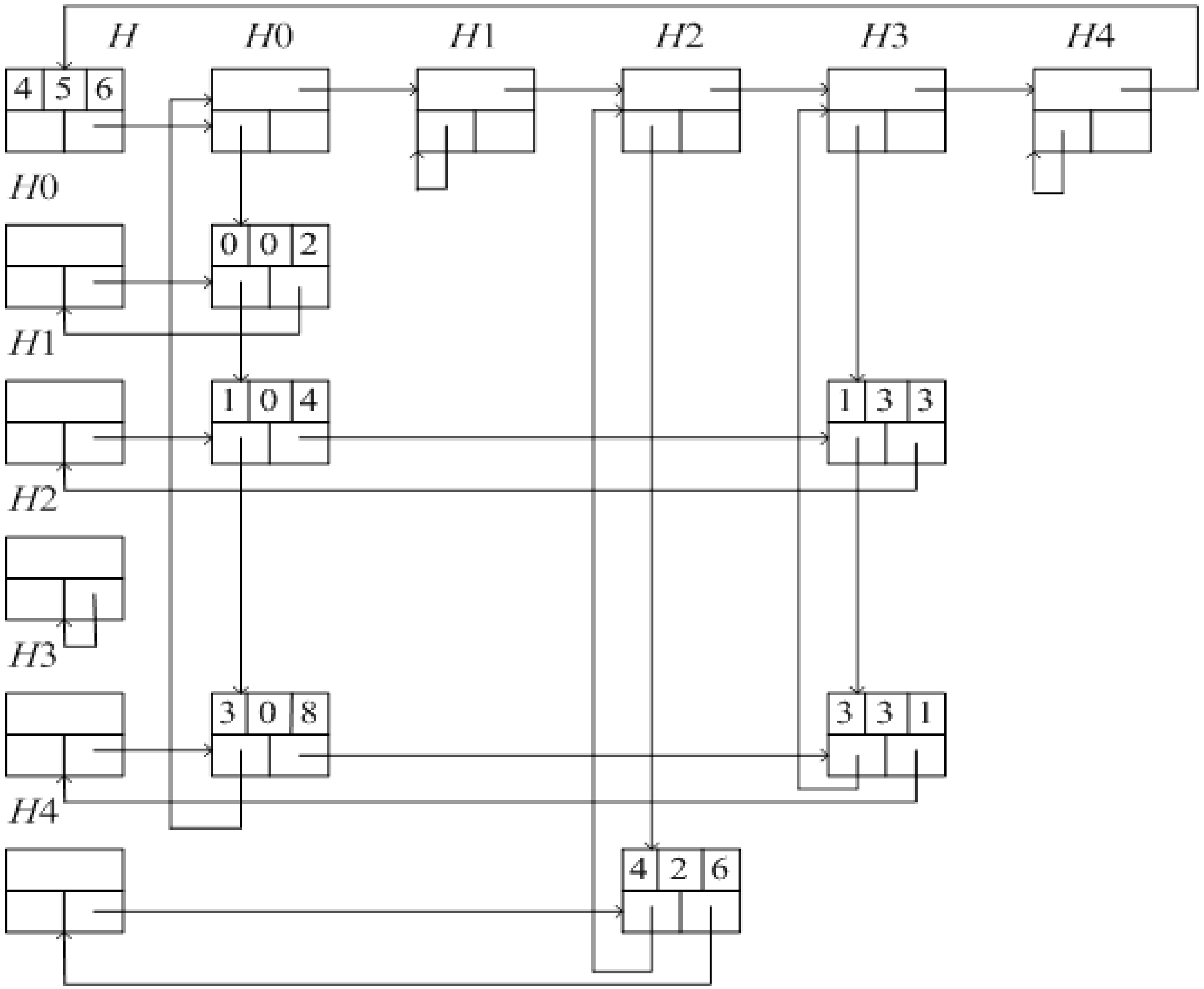
```
typedef struct matrixNode {
	matrixPointer down;
	matrixPointer right;
	tagfield tag;
	union
	{
	matrixPointer next;
	entryNode entry;
	} u;
};
```

Example

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$

DATA STRUCTURES AND ITS APPLICATIONS

Sparse Matrix Representation: Linked representation



Courtesy: “Fundamentals of Data Structures” By Ellis Horowitz and Sartaj Sahni

Sparse Matrix Representation Summary

Sparse matrix representation

- Triple
- Linked Representation

Concepts can be applied to implement the following operations

- Create_SparseMatrix()
- Transpose_of_SparseMatrix()
- Add_SparseMatrices()
- Multiple_SparseMatrices()

Prof. Vandana M L
Department of Computer Science and Engineering

vandanamd@pes.edu

Data Structures and its Applications

Case Study – Text Editors

Dinesh Singh

Department of Computer Science and Engineering

Text Editor – An Introduction

- Editors or text editors are software programs that enable the user to create and edit text files.

- In the field of programming, the term editor usually refers to source code editors that include many special features for writing and editing code.

- Notepad++, Wordpad, vi, emacs, Jed, pico are some of the editors used on Windows, linux, UNIX OS.

- Features normally associated with text editors are :
  Storage of data, Cursor Movement.
  Insertion, Deletion, Find & Replace, Cut, Copy , paste of text , Saving, etc.

Types of Editors

There are generally five types of editors as described below:

- **Line editor**: Can edit only one line at a time or an integral number of lines. You cannot have a free-flowing sequence of characters. It will take care of only one line.
  Ex : Teleprinter, edlin, teco

- **Stream editors**: In this type of editors, the file is treated as continuous flow or sequence of characters instead of line numbers, which means here you can type paragraphs.
  Ex : Sed editor in UNIX

Text Editor – An Introduction

- **Screen editors**: In this type of editors, the user is able to see the cursor on the screen and can make a copy, cut, paste operation easily. It is very easy to use mouse pointer.
  Ex : vi, emacs, Notepad, Notepad++

- **Word Processor**: Overcoming the limitations of screen editors, it allows one to use some format to insert images, files, videos, use font, size, style features.
  Focuses on Natural language.

- **Structure Editor**: Structure editor focuses on programming languages. It provides features to write and edit source code.
  Ex : Netbeans IDE, gEdit, Notepad++

Editing Process:

- Editor program enables is used to create, edit and modify a document.
- A document may include some images, files, text, equations, and diagrams as well.
- Limited only to text editors - character strings.

Design of a Text Editor

The document editing process mainly comprises of the following four tasks :

- The part of the document to be edited or modified is selected.
- Determining how to format these lines on view and how to display it.
- Specify and execute the operations that modify the document.
- Update the view properly.

Design of a Text Editor

The text editing process include the following steps:

- Formatting : Visibility on display screen.
- Filtering : Finding out the main/important subset.
- Traveling : Locating the area of interest

User Interface of editors:

- The user interface of editors typically means the input, output and the interaction language.
- The input devices are used to enter text, data into a document or to process commands.
- The output devices are used to display the edited form of the document and the results of the operation/commands executed.
- The interaction language provides the interaction with the editor.

1. *Input Devices :*

- Input devices are generally divided as text input, button devices and locator devices.
- Text device is a keyboard. Button devices are special function keys.
- The locator devices include the mouse.
- There are special voice devices as well which writes into text whatever you speak.

Design of a Text Editor

2. Output Devices :

- TFT monitors,
- Printers,
- Teletypewriters,
- Cathode ray tube technology,
- Advanced CRT terminals.

3. *Interaction language :*

- The interaction language could be, typing oriented or text command-oriented or could be menu oriented user interface as well.

- Typing or text command-oriented interaction language is very old used with the oldest editors, in the form of commands, use of functions and control keys etc.

- Menu oriented interface has a menu with the set of multiple choice of text strings. The display area is limited and the menus can be turned on/off by the user.

Objectives:

- Identify the character typed or pressed by the user on the keyboard.
- Store the character in the buffer.
- Display the character on the screen / display device in the user readable form.
- Update the display position (cursor).
- Provide opportunity to the user to modify the document.

How to read a character, store, recognize and display data (Character) on the standard output device?

For every character entered by the user,

- The data input key such as an alphabet, number or a symbol, is stored in the buffer.
- Buffer can be realized using any data structure.
  - The data structure could be an Array, Linked List, Tree, etc.
- The data from the buffer is read and displayed at the present cursor position.
- The cursor position is moved one step forward and displayed again.
- This process is continued as long as the EOF is read from the user.
- This is the front end story of the editor.

Text Editor – Front end :

Case 1: Create a new file, add or append the text and display on the screen.

Initially, an empty screen is displayed as shown below.

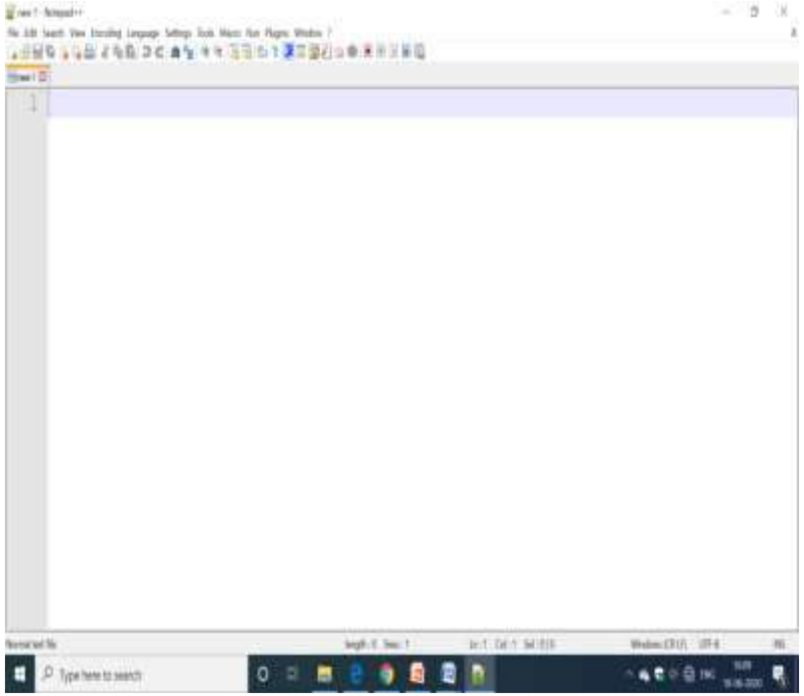


Fig A: Notepad++

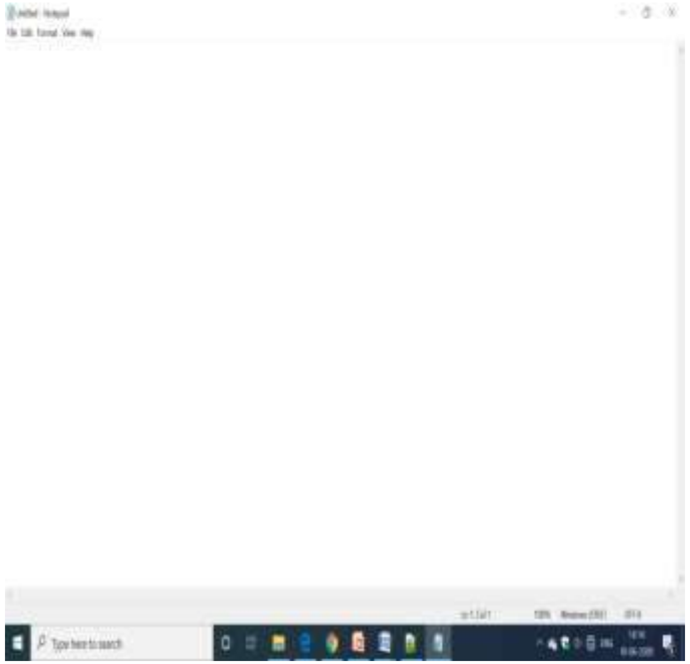
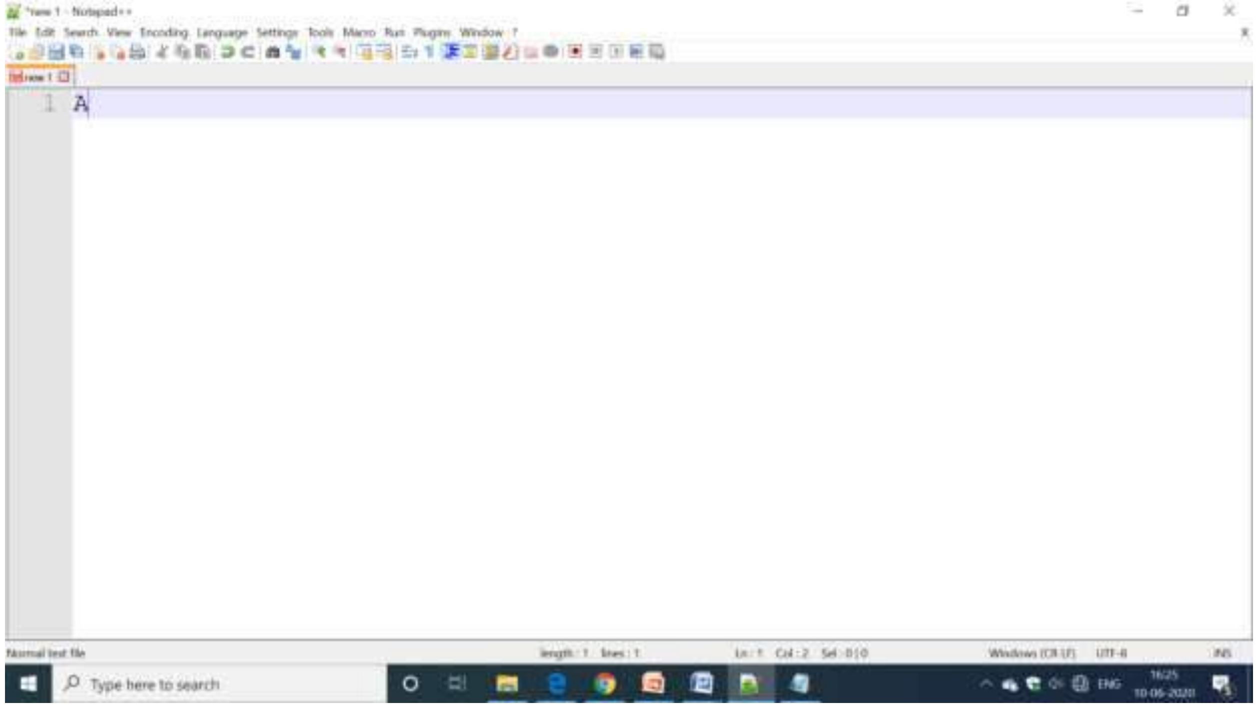


Fig B: Notepad

Can be any other editor as well.

Text Editor – Front end :

An entry of data(character) from the input device(keyboard), gets display on the screen (shown).



What happens at the backend ?

Text Editor : Back End Story - What happens at the backend ?

- A character typed is stored in the data structure created.

Case Study: Data Structure used is linked list.

- Linked list is currently pointing to NULL.
- Indicates no data typed.

Text Editor : Back End Story - What happens at the backend ?

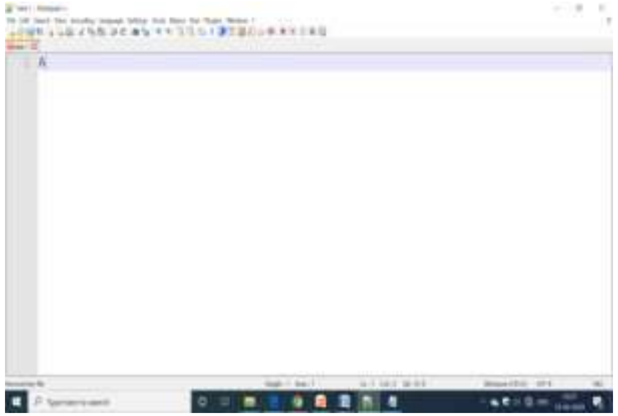
- A character typed is stored in the data structure created.

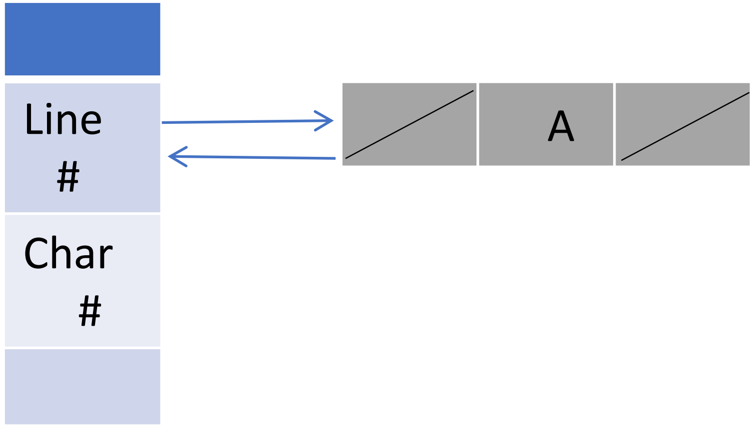
Case Study: Data Structure used is linked list.

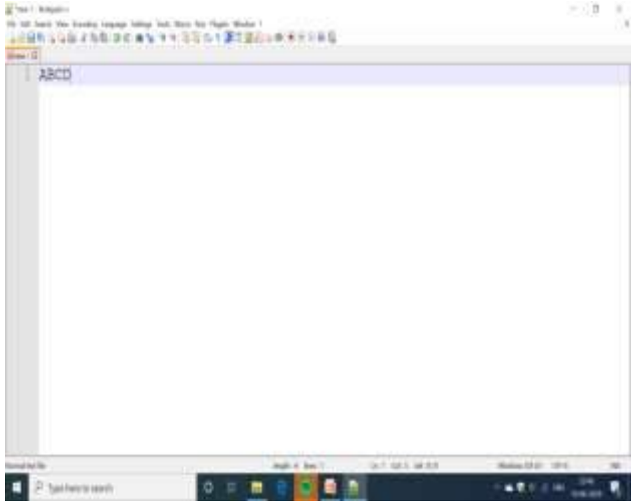
- Linked list is currently pointing to NULL.
- Indicates no data typed.

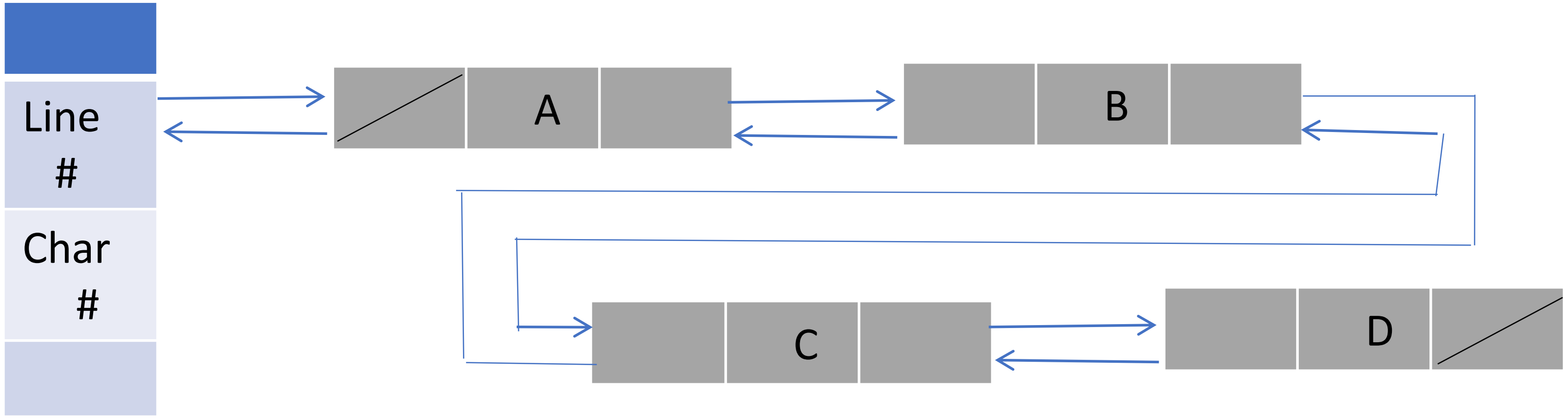
Entering A and the next characters the structure looks like…

Implementation using Doubly Linked List

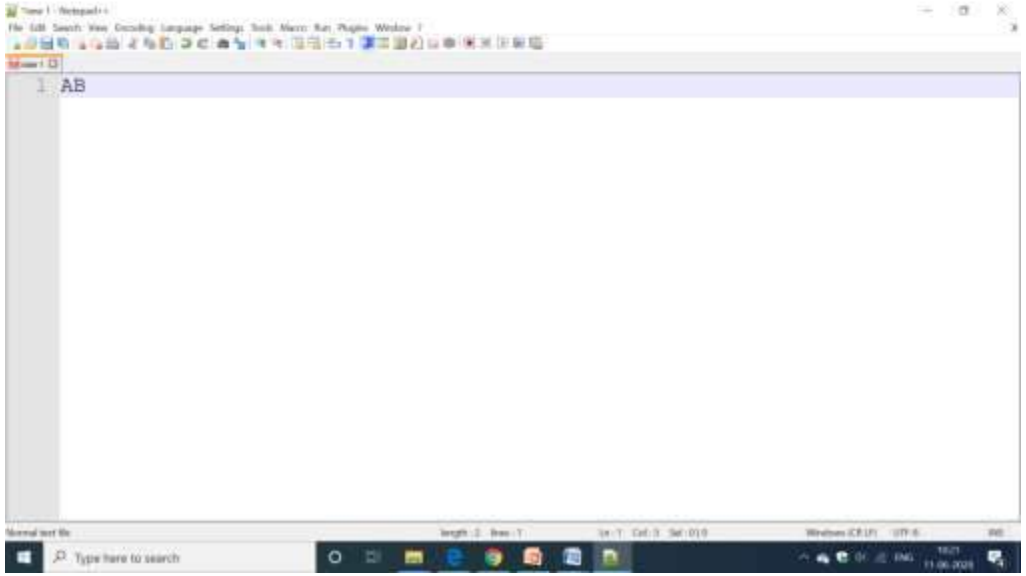


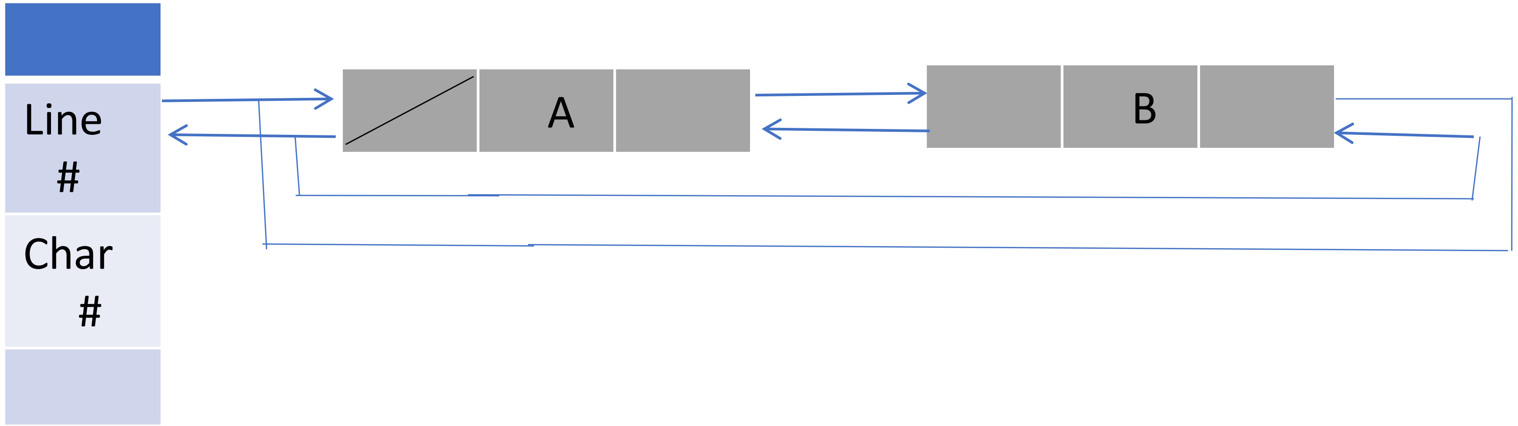






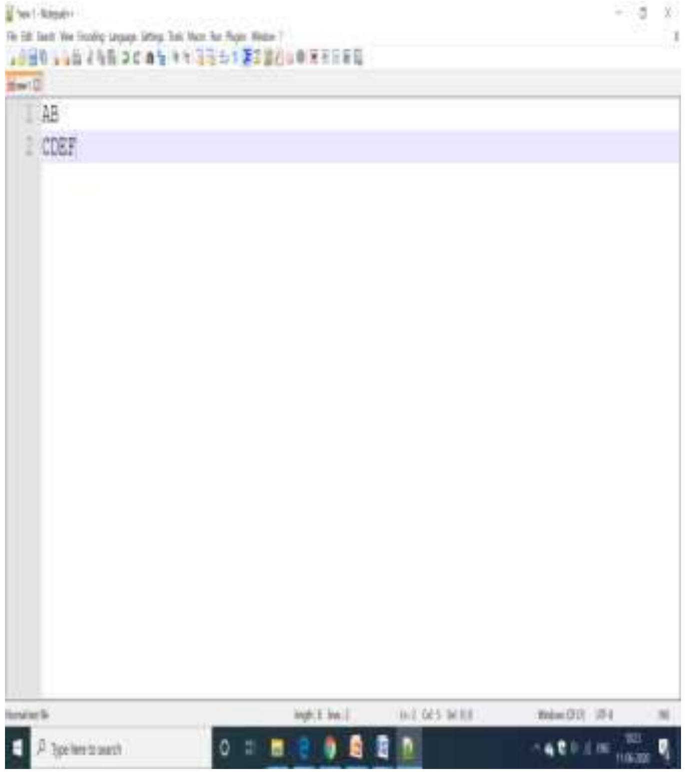
Implementation using Doubly Linked Circular List

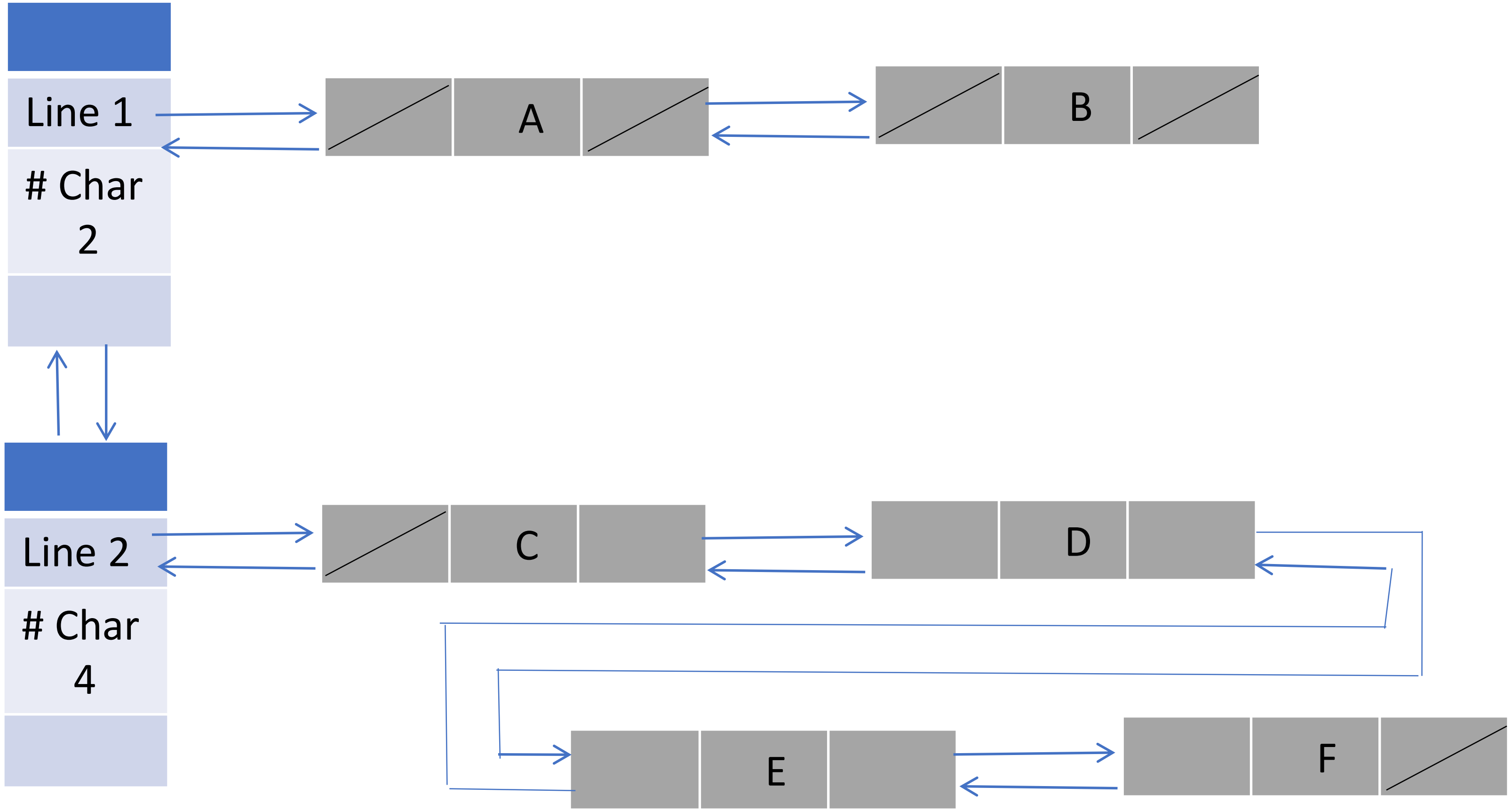




Data Structures and its Applications

Multiple lines - Implementation using Doubly Linked List





THANK YOU

Dinesh Singh
Department of Computer Science & Engineering

badriprasad@pes.edu

+91 80 26721983 Extn 721

Data Structures and its Applications

Case Study – Symbol Tables

Dinesh Singh
Department of Computer Science and Engineering

Assembler – An Introduction

- Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.
- It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code.
- Types of Assemblers
  - Single Pass Assembler : the above task is done in one pass
  - Two Pass Assembler : the above task is done in two passes.

Pass-1:

Define symbols and literals and remember them in symbol table and literal table respectively.

Keep track of location counter

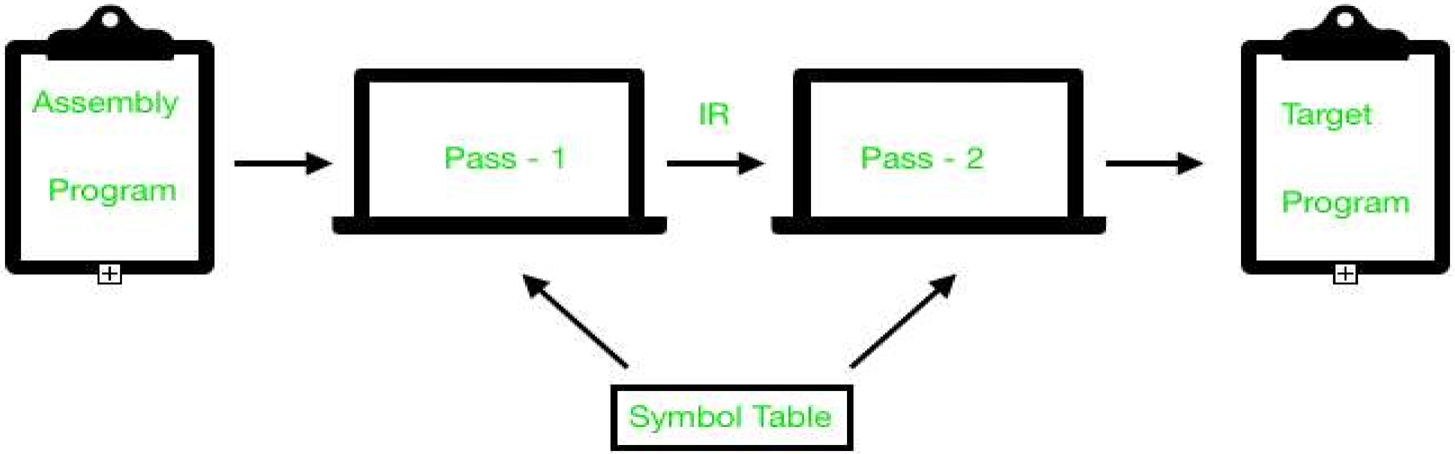
Process pseudo-operations

Pass-2:

Generate object code by converting symbolic op-code into respective numeric op-code

Generate data for literals and look for values of symbols

Assembler – An Introduction



Items stored in Symbol table:

- •Variable names and constants
- •Procedure and function names
- •Literal constants and strings
- •Labels

Information used by assembler from Symbol table:

- •Data type and name
- •Declaring procedures
- •Offset in storage
- •If structure or record then, pointer to structure table.
- •For parameters, whether parameter passing by value or by reference
- •Number and type of arguments passed to function
- •Base Address

Symbol Table

Following are commonly used data structure for implementing symbol table :-

List –

- In this method, an array is used to store names and associated information.
- New names are added in the order as they arrive
- To search for a name we start from beginning of list till available pointer and if not found we get an error “use of undeclared name”
- While inserting a new name we must ensure that it is not already present otherwise error occurs.
- Insertion is fast , but lookup is slow for large tables .
- Advantage is that it takes minimum amount of space.

Linked List

This implementation is using linked list. A link field is added to each record.

- Searching of names is done in order pointed by link of link field.
- A pointer "First" is maintained to point to first record of symbol table.
- Insertion is fast , but lookup is slow for large tables

Other Data Structures

- Hash table.
- Binary Search Tree.

Operations of Symbol table –

The basic operations defined on a symbol table include:

Operation	Function
allocate	Allocate new empty symbol table
Free	To remove all entries and free storage of symbol table
Look up	Search for a name and return pointer to its entry
Insert	To insert a name in the symbol table and return pointer to its entry
Set attribute	To associate an attribute with a given entry
Get attribute	To get an attribute associated with a given entry

THANK YOU

Dinesh Singh
Department of Computer Science & Engineering

badriprasad@pes.edu

+91 80 26721983 Extn 721