DATA STRUC

Vandana M L
Department of C

DATA STRUCTURES AND ITS A

Singly Linked List

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Singly Linked List Operations

Deleting a node

There are 3 cases

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

DATA STRUCTURES AND ITS APPLICATIONS
Singly Linked List Operations

Deleting first 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

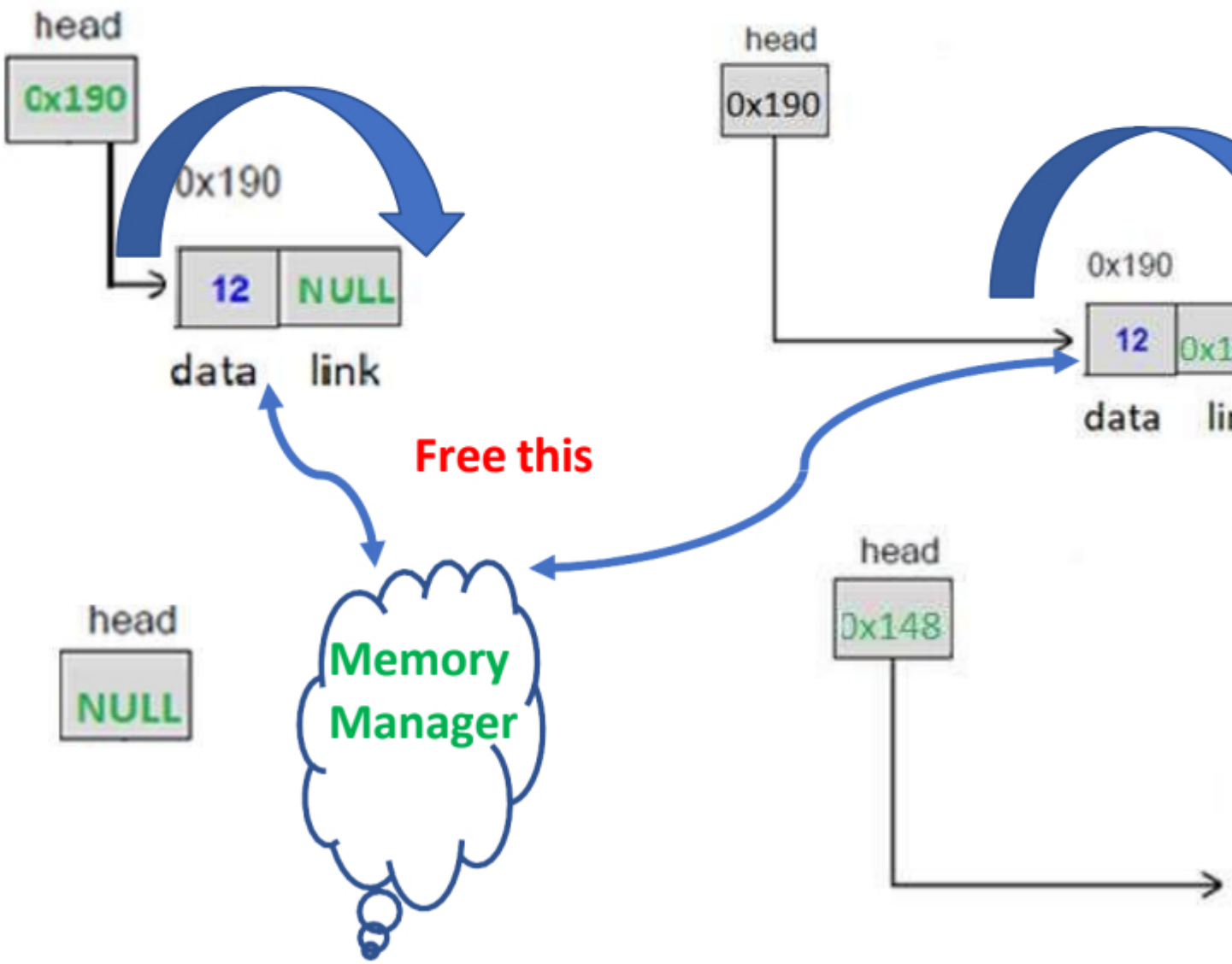
- Change head to point to second node
- Delete the first node

DATA STRUCTURES AND ITS APPLICATIONS

Singly Linked List Operations

Deleting first node

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



Deleting last node

Case 1: Linked list is empty

Case 2: Linked list with a single node

- delete the node
- set head to NULL

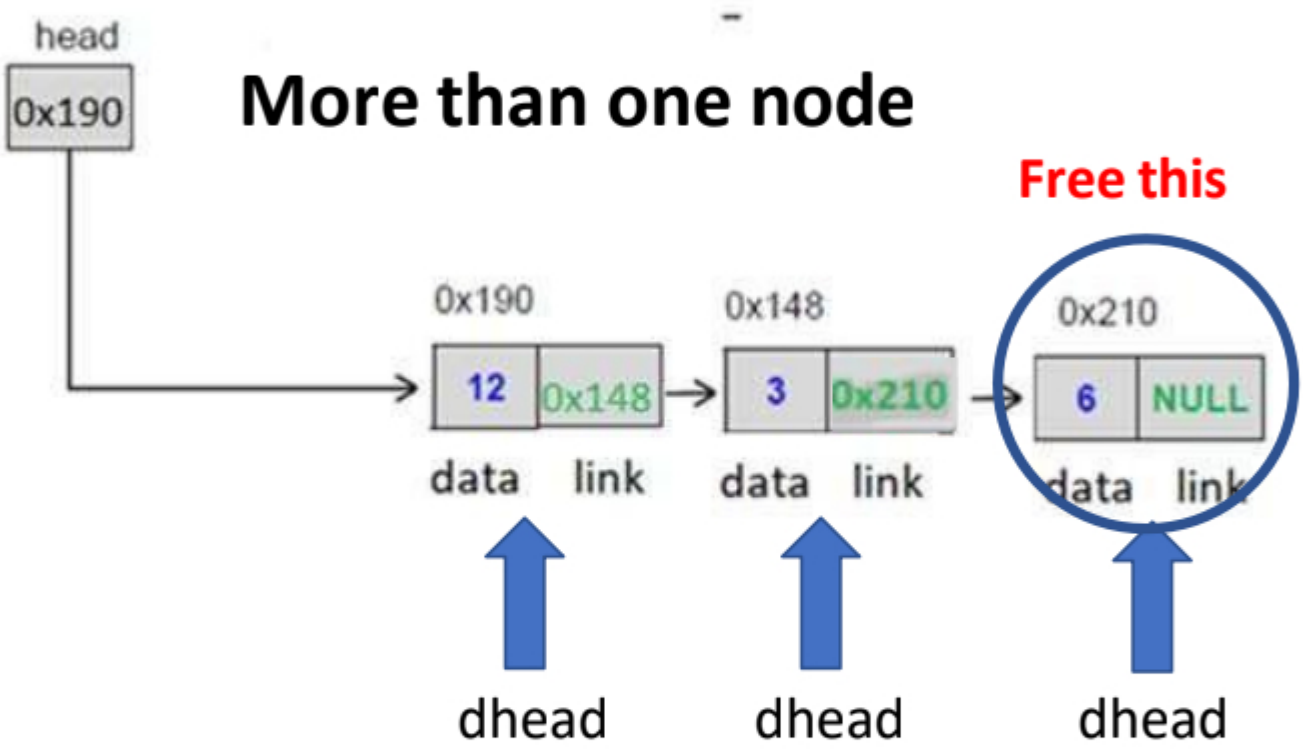
Case3:Linked list has more than one node

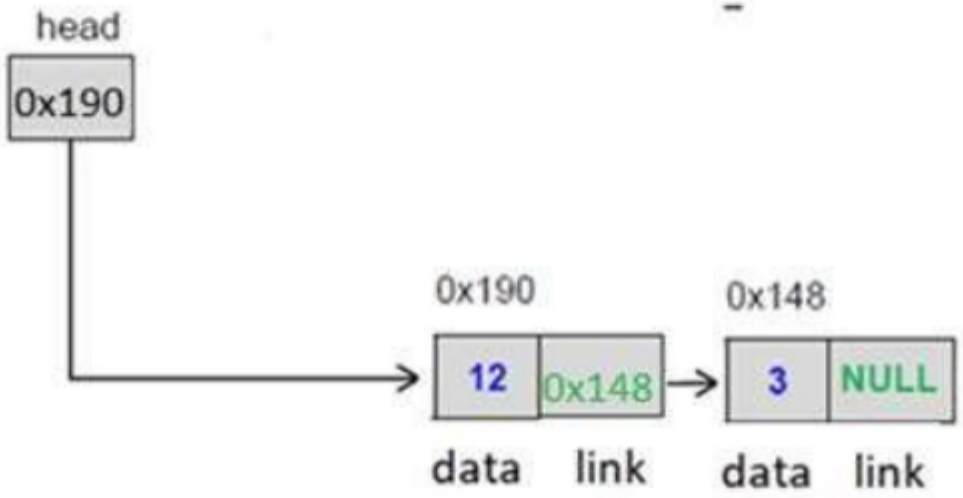
- Traverse the linked list to point to second node
- Delete the last node
- Set link field of second last node to NULL

DATA STRUCTURES AND ITS APPLICATIONS

Singly Linked List Operations

Deleting last node





DATA STRUCTURES AND ITS APPLICATIONS
Singly Linked List Operations

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

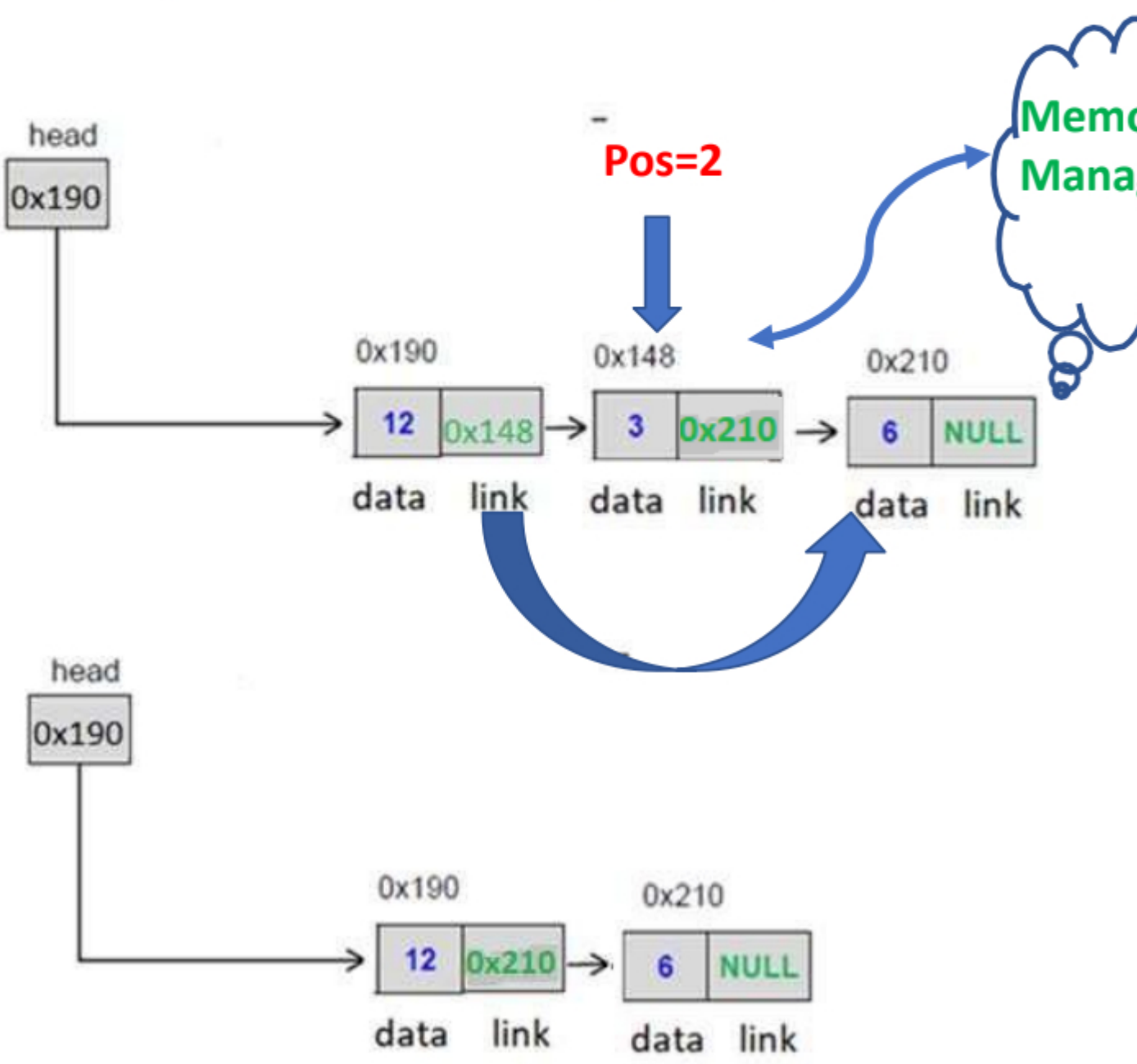
- Traverse linked list to get the desired
- keep track of previous node
- set previous node link field to link fie current node
- delete the current node

Else

- print invalid position

DATA STRUCTURES AND ITS APPLICATIONS
Singly Linked List Operations

Deleting node from a given position



DATA STRUCTURES AND ITS APPLICATIONS
Lecture Summary

Singly Linked List delete operation

Apply the concepts to implement following o

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

DATA STRUCTURES AND ITS APPLICATIONS

Multiple-Choice-Questions (MCQ's)

1. Which is the correct sequence of op
head node of an SLL?

a) head = head->next; free(head);

b) temp = head; head = head->next; free(

c) free(head); head = head->next;

d) head->next = head; free(head);

DATA STRUCTURES AND ITS APPLICATIONS

Multiple-Choice-Questions (MCQ's)

1. Which is the correct sequence of op
head node of an SLL?

a) head = head->next; free(head);

b) temp = head; head = head->next; free(

c) free(head); head = head->next;

d) head->next = head; free(head);

2. To delete the node after a given node
correct?

a) p->next = p->next->next; free(p);

b) temp = p->next; p->next = temp->next;

c) temp = p; p = p->next; free(temp);

d) free(p->next); p->next = p->next->next;

2. To delete the node after a given node
correct?

a) p->next = p->next->next; free(p);

b) temp = p->next; p->next = temp->next;

c) temp = p; p = p->next; free(temp);

d) free(p->next); p->next = p->next->next;

3. When deleting the last node in an Sl what is required?

a) Traverse to the second-last node, set i the last node.

b) Set head = NULL directly.

c) Free the last node and leave the secon

d) No traversal is required.

3. When deleting the last node in an Sl what is required?

a) Traverse to the second-last node, set i the last node.

b) Set head = NULL directly.

c) Free the last node and leave the secon

d) No traversal is required.

4. What happens if you attempt to delet

a) The program executes without issue.

b) A segmentation fault occurs due to NUL

c) The head pointer is automatically initiali

d) A new node is created.

4. What happens if you attempt to delet

a) The program executes without issue.

b) A segmentation fault occurs due to NUL

c) The head pointer is automatically initiali

d) A new node is created.

5. To delete a node at position n in an which of the following is correct?

a) Traverse n nodes and free the node at

b) Traverse n-1 nodes to find the previou its next and free the target node.

c) Swap data of the nth and (n-1)th node node.

d) Always delete from the head if n > 1.

5. To delete a node at position n in an which of the following is correct?

a) Traverse n nodes and free the node at

b) Traverse n-1 nodes to find the previou its next and free the target node.

c) Swap data of the nth and (n-1)th node node.

d) Always delete from the head if n > 1.

THANK Y

Vandana M L
Department o

vandanamd@

DATA STRU

Vandana M L

Department of C

DATA STRUCTURES AND ITS A

Singly Linked List

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Singly Linked List: Node Structure

Defining node structure

```
struct node
{
int data;
struct node *link;
};

typedef struct node NODE;
```


```
struct polyc
{
  int coeff;
  int expo;
};

typedef stru
 {
polydata da
struct node
}polynode;
```

DATA STRUCTURES AND ITS APPLICATIONS

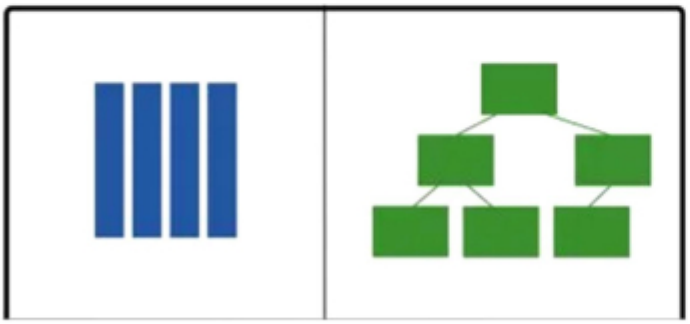
Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Introduction to Data Structures

Vandana M L
Department of Computer Science and Engineering

Data Structure is a scheme of organizing data in the memory of the computer in such a way that various operations can be performed efficiently on this data



Why Data Structure?

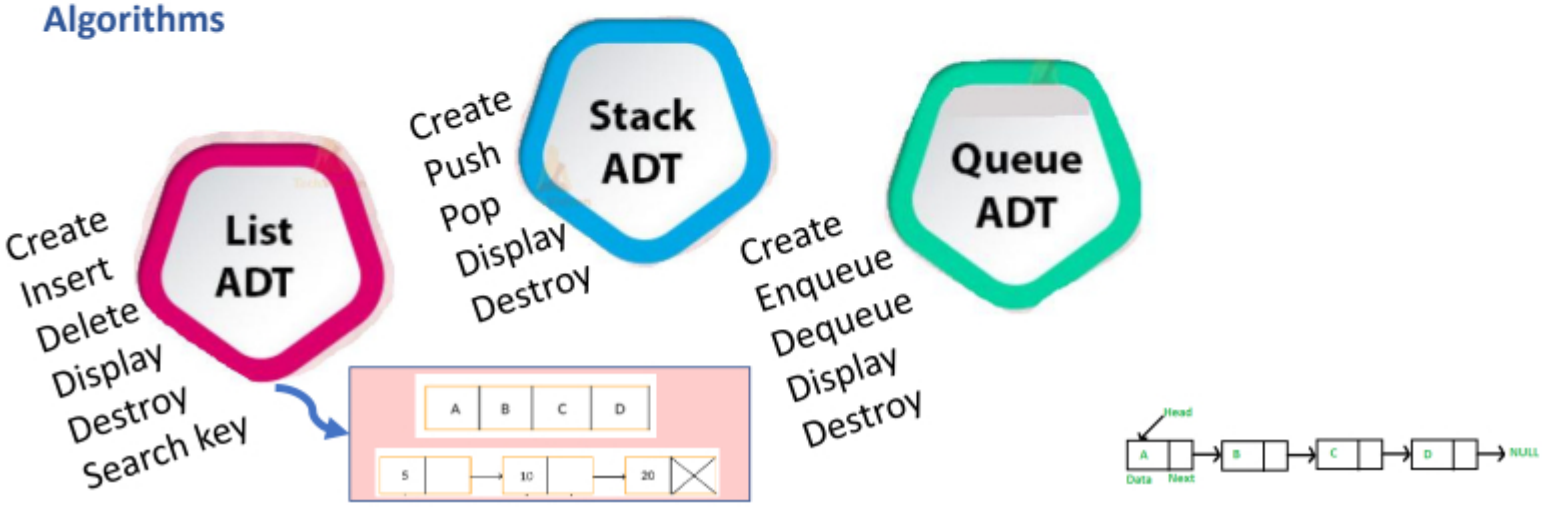
Why Data Structures?

- Computer systems deal with large amount of data (text ,image, relational data etc.)
- Data is just the raw material for information, analytics, business intelligence, advertising, etc.
- The way data is organized in memory plays a key role in deciding the time complexity of the algorithms designed for solving the problems
- Data Structures and algorithm go hand in hand

Importance of Data Structures

- Data Structures is most fundamental and building block concept in computer science
- Good knowledge of Data Structures is required to build efficient software systems

- Abstract Data Type is used to represent data and operations associated with an entity from the point of view of user irrespective of implementation
- ADT can be implemented using one or more **Data Structures** and **Algorithms**



Classification of Data Structures

- **Linear Data Structures**
  Stack, Queue, Linked List

- **Non Linear Data Structures**
  Tree , Graph

Linear Organisation

Stack
Queue
Linked List
Linear List using Array

Non Linear Organisation

Tree
Graph

Few Applications of Linear Data Structures

- **Array**
  - To implement other data structures
  - To store files in memory
- **Linked Lists**
  - To implement other data structures
  - To manipulate large numbers
- **Stacks**
  - Recursion
  - Infix to postfix conversion
- **Queues**
  - Process Scheduling
  - Event handling

Few Applications of Non Linear Data Structures

- **Tree**
  - Auto complete features (Trie)
  - Used by operating systems to maintain the structure of a file system
- **Heaps**
  - Priority Queue implementation
  - Heap Sort
- **Graphs**
  - Computer Networks
  - Shortest Path Problems

Unit -1 : Linked List and Stacks

Review of C , Static and Dynamic Memory Allocation. Linked List: Doubly Linked List, Circular Linked List – Single and Double, Multilist: Introduction to sparse matrix (structure). Skip list Case study: Dictionary implementation using skip list Stacks: Basic structure of a Stack, Implementation of a Stack using Arrays & Linked list. Applications of Stack: Function execution, Nested functions, Recursion: Tower of Hanoi. Conversion & Evaluation of an expression: Infix to postfix, Infix to prefix, Evaluation of an Expression, Matching of Parenthesis.

Unit -2 : Queues and Trees

Queues & Dequeue: Basic Structure of a Simple Queue, Circular Queue, Priority Queue, Dequeue and its implementation using Arrays and Linked List. Applications of Queue: Case Study – Josephus problem, CPU scheduling- Implementation using queue (simple /circular). General: N-ary trees, Binary Trees, Binary Search Trees (BST) and Forest: definition, properties, conversion of an N-ary tree and a Forest to a binary tree. Traversal of trees: Preorder, Inorder and Postorder.

Unit -3 : Application of Trees and Introduction to Graphs

Implementation of BST using arrays and dynamic allocation: Insertion and deletion operations, Implementation of binary expression tree., Threaded binary search tree and its implementation. Heap: Implementation using arrays. Implementation of Priority Queue using heap - min and max heap. Applications of Trees and Heaps: Implementation of a dictionary / decision tree (Words with their meanings). Balanced Trees: definition, AVL Trees, Rotation, Splay Tree, Graphs: Introduction, Properties, Representation of graphs: Adjacency matrix, Adjacency list. Implementation of graphs using adjacency matrix and lists. Graph traversal methods: Depth first search, Breadth first search techniques. Application: Graph representation: Representation of computer network topology.

Unit -4 : Applications of Graphs , B-Trees, Suffix Tree and Hashing

Application of BFS and DFS: Connectivity of graph, finding path in a network. Suffix Trees: Definition, Introduction of Trie Trees, Suffix trees. Implementations of TRIE trees, insert, delete and search operations. Hashing: Simple mapping / Hashing: hash function, hash table, Collision Handling: Separate Chaining & Open Addressing, Double Hashing, and Rehashing. Applications: URLs decoding, Word prediction using TRIE trees / Suffix Trees.

Text Book :

"Data Structures using C / C++", Langsum Yedidyah, Moshe J Augenstein, Aaron M Tenenbaum Pearson Education Inc, 2nd edition,2015.

Reference Book:

"Data Structures and Program Design in C", Robert Kruse, Bruce Leung, C.L Tondo, Shashi Mogalla, Pearson, 2nd Edition, 2019

Data Structures and its Applications

Evaluation Policy

Pattern:	ISA: 50 marks ESA: 50 marks
ISA 1 / ISA 2:	Hybrid / CBT, conducted for 40 marks, scaled down to 20 marks (each) (16 * 1) mark MCQs + (4 * 2) marks MCQs + (4 * 4) marks descriptive questions. **Total: 20 + 20 marks.**
Lab Component/FSA:	10 Labs (Implementation based): 10 marks Jackfruit problem (Mini): 10 marks **Total : 20 Marks**
Assignment / Experiential Learning	Banana problem: 5 marks (pen and paper) (classnotes , HW) Orange problem: 5 marks (Implementation based in assessment center) **Total: 10 marks**
ESA:	25 marks from each unit, conducted for 100 marks, reduced to **50 marks**
ISA + ESA + Lab + Experiential Learning	120 reduced to **100 marks**

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu

DATA STRU

Vandana M

Departmer

DATA STRUCTURES AND ITS APP

Memory Allocation

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Memory Allocation

- Static Memory Allocation
- Dynamic Memory Allocation

DATA STRUCTURES AND ITS APPLICATIONS

Static Memory Allocation

- allocated by the compiler.
- Exact size and type of memory must be k
- Memory is allocated in stack area

```
int b;

int c[10] ;
```

DATA STRUCTURES AND ITS APPLICATIONS

Disadvantages of Static Memory Allocation

- Memory allocated can not be altered dur
  is allocated during compile time
- This may lead to under utilization or over
  memory
- Memory can not be deleted explicitly onl
  be overwritten
- Useful only when data size is fixed and kn
  processing

DATA STRUCTURES AND ITS APPLICATIONS

Dynamic Memory Allocation

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

Dynamic Memory Allocation Functions:

- Allocate memory - malloc(), calloc(), a
- Free memory - free()

Dynamic Memory Allocation Functions: mall

To allocate memory use

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

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

Dynamic Memory Allocation Functions: mall

To allocate space for 100 integers:

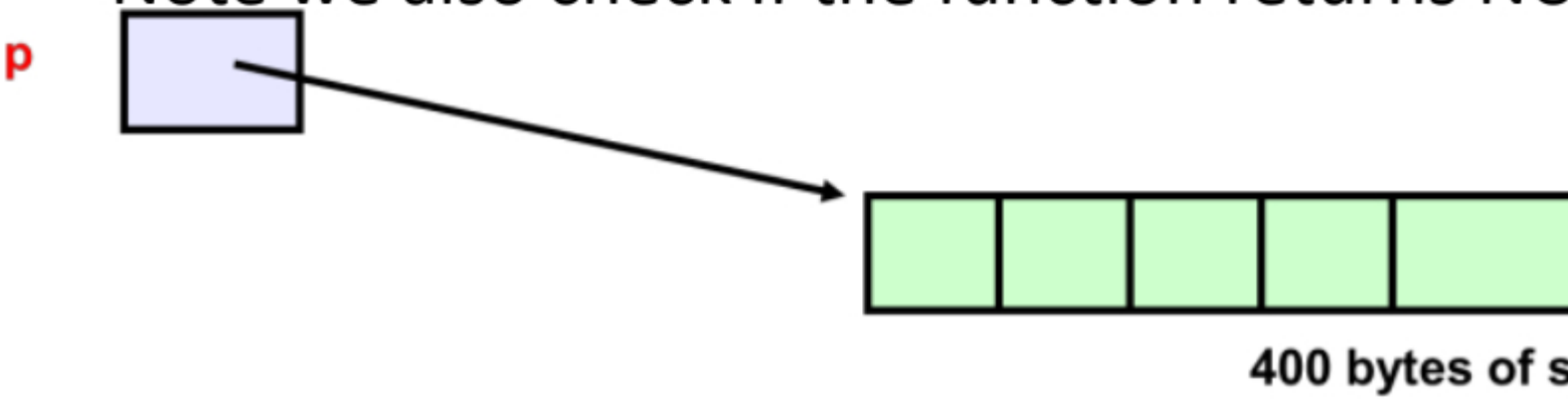
```
int *p;

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 NU



Dynamic Memory Allocation Functions: mall

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

  Allocates 20 bytes of space for the point

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

  Allocates space for a structure array of 10
  to a structure element of type struct stu

Always use sizeof operator to find number o
as it can vary from machine to machine

Dynamic Memory Allocation Functions: mall

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

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

The n integers allocated can be accessed as *p, just as p[0], p[1], p[2], …,p[n-1]

Dynamic Memory Allocation Functions: free

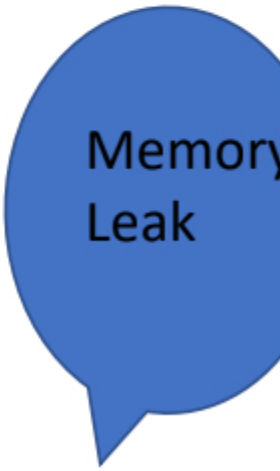
To release allocated memory use

```
free(ptrvariable)
```

- Deallocates memory allocated by mallo
- Takes a pointer as an argument.

e.g

```
free(newPtr);
```



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
else it'll return a NULL pointer.

Dynamic Memory Allocation Functions: reall

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));
```

DATA STRUCTURES AND ITS APPLICATIONS

Lecture Summary

Memory Allocation

- Static Memory allocation
- Dynamic memory allocation

Apply the concepts to implement C program f

- Multiply two matrices . Allocate the me

DATA STRUCTURES AND ITS APPLICATIONS

Multiple-Choice-Questions (MCQ'S)

1. Which of the following is true ab

a) Memory is allocated during runtime

b) The size of memory block can be a

c) Memory is allocated during compile

d) It uses malloc() and free() function

DATA STRUCTURES AND ITS APPLICATIONS

Multiple-Choice-Questions (MCQ'S)

1. Which of the following is true ab

a) Memory is allocated during runtime

b) The size of memory block can be a

c) Memory is allocated during compil

d) It uses malloc() and free() function

2. In C, which of the following dynar memory that is not initialized?

a) malloc()

b) calloc()

c) realloc()

d) free()

2. In C, which of the following dynar memory that is not initialized?

a) malloc()

b) calloc()

c) realloc()

d) free()

3. Which of the following statement
memory allocation?

a) It is performed at runtime.

b) It uses heap memory.

c) The size of memory block cannot be

d) realloc() can resize an already alloc

3. Which of the following statement
memory allocation?

a) It is performed at runtime.

b) It uses heap memory.

c) The size of memory block cannot b

d) realloc() can resize an already alloc

4. What will happen if free() is not ca
allocated memory?

a) The program will not compile.

b) Memory leak will occur.

c) The stack will overflow.

d) The data will be automatically delete

4. What will happen if free() is not ca
allocated memory?

a) The program will not compile.

b) Memory leak will occur.

c) The stack will overflow.

d) The data will be automatically delete

5. Which of the following is incorrect?

a) Static allocation uses stack, dynamic

b) Static allocation is determined at comp

c) Static memory can be resized using re

d) Dynamic memory requires explicit allo

5. Which of the following is incorrect?

a) Static allocation uses stack, dynamic

b) Static allocation is determined at comp

c) Static memory can be resized using re

d) Dynamic memory requires explicit allo

THANK Y

Vandana M L

Department o

vandanamd@

+91 7411716

PES UNIVERSITY

DATA STRUC

Vandana M L

Department of Co

DATA STRUCTURES AND ITS A

Introduction to Singly Linked

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

List Data Structure

List

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

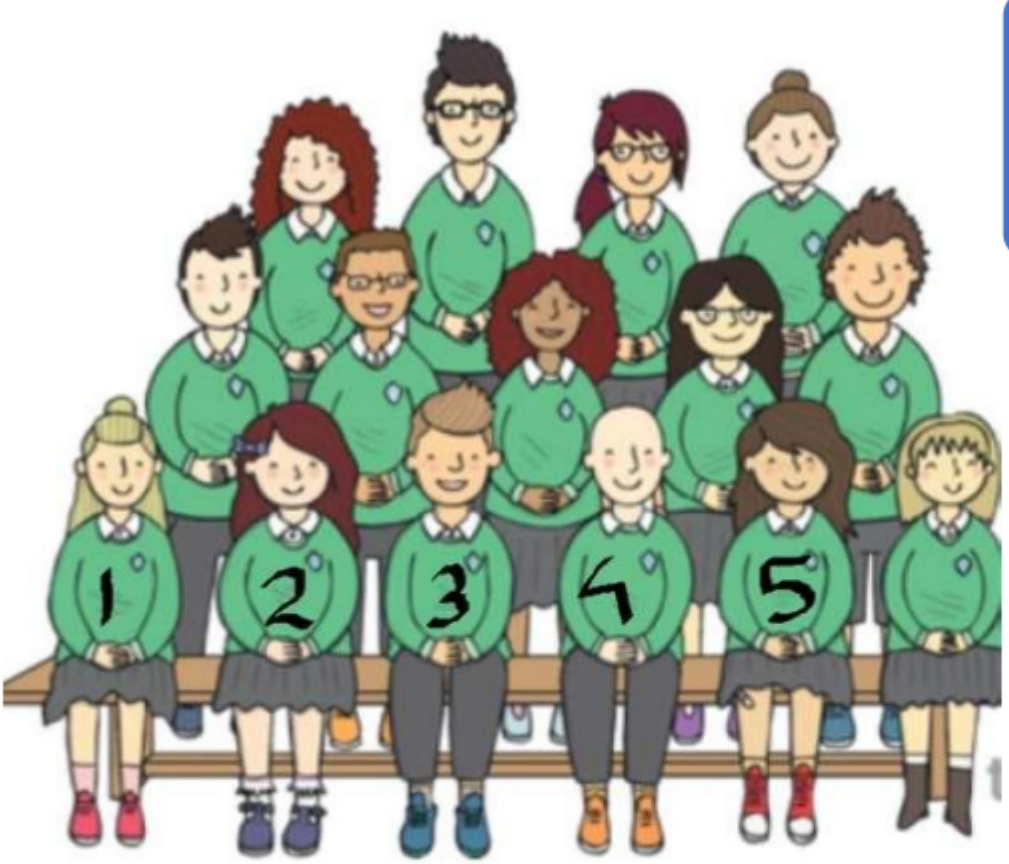
DATA STRUCTURES AND ITS APPLICATIONS

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

DATA STRUCTURES AND ITS APPLICATIONS

Understanding Array List (Linear List using A



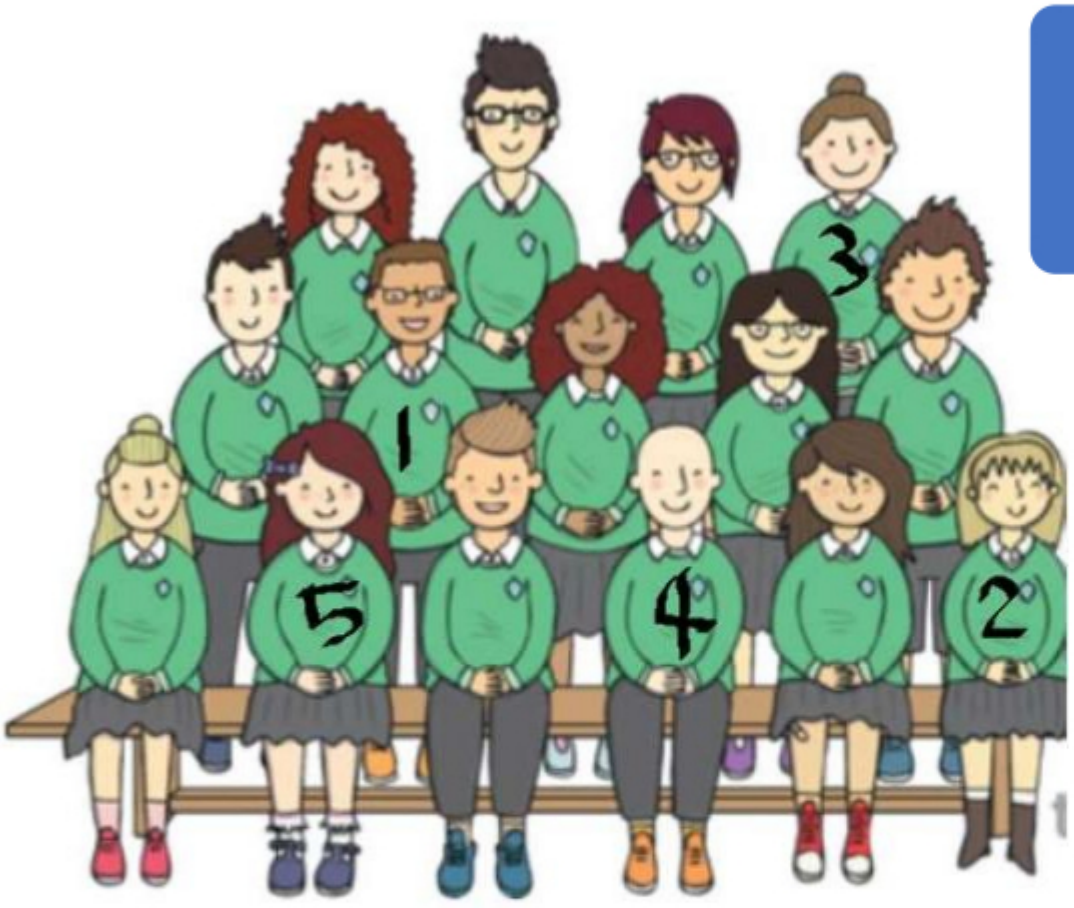
Placement

- Sequenti

Arr

DATA STRUCTURES AND ITS APPLICATIONS

Understanding Linked List



Placement

- Random

Link

DATA STRUCTURES AND ITS APPLICATIONS

Array List Vs Linked List

ArrayList	
Fixed size: Resizing is expensive	Dynamic si
Insertions and Deletions are inefficient	Insertions efficient
Elements in contiguous memory locations	Elements r memory lo
May result in memory wasteage if all the allocated space is not used	Since mem dynamicall there is no
Sequential and random access is faster	Sequential slow

DATA STRUCTURES AND ITS APPLICATIONS

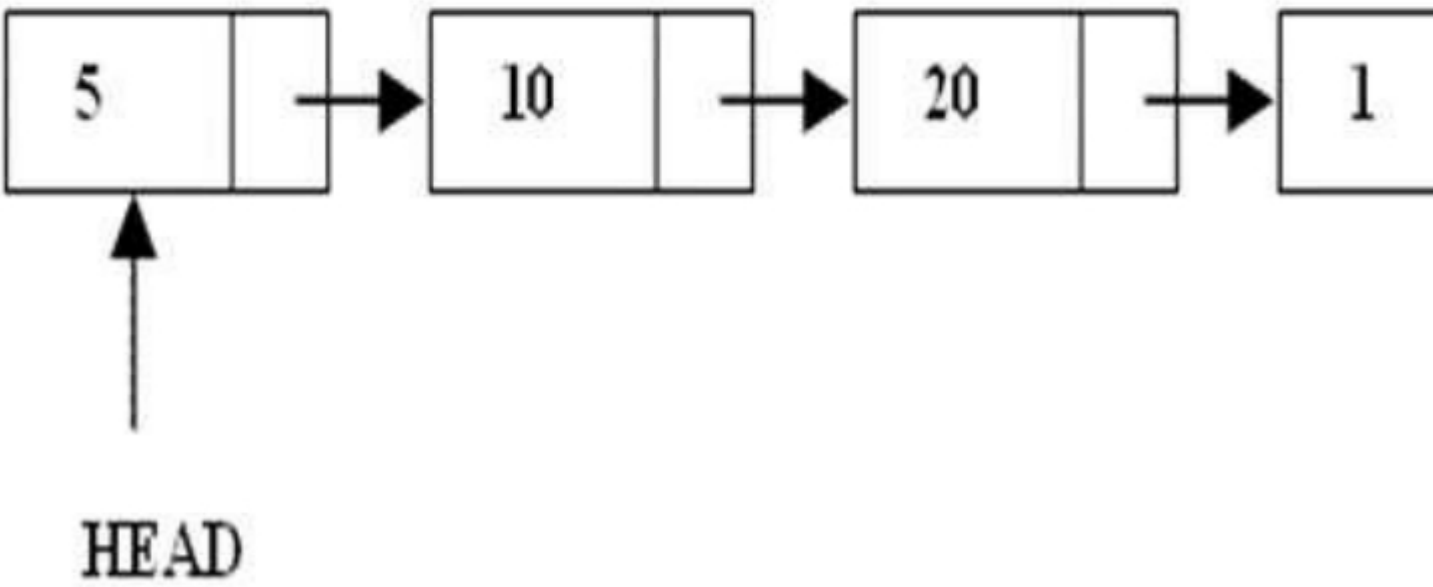
Types of Linked List

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

DATA STRUCTURES AND ITS APPLICATIONS

Singly Linked List

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



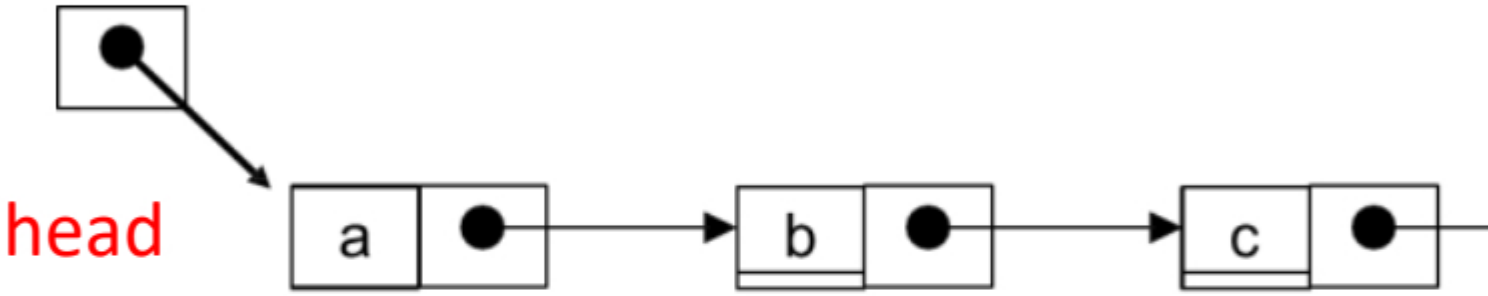
DATA STRUCTURES AND ITS APPLICATIONS

Single Linked List

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

DATA STRUCTURES AND ITS APPLICATIONS

Single Linked List : Schematic Representation



- Each node contains a value(data) and a point
node in the list

- Head/start is a pointer which points at the fi
list

DATA STRUCTURES AND ITS APPLICATIONS

Lecture Summary

Singly Linked List

Apply the concepts to answer the following q

- Give structure definition for node of si
(employee no , name, salary ,designation

DATA STRUCTURES AND ITS APPLICATIONS

Multiple-Choice-Questions (MCQ's)

1. Which of the following best descri
nodes in an SLL?

a) Continuous memory blocks are used

b) Nodes are allocated dynamically and

c) Memory allocation is fixed during cor

d) Memory blocks are always allocated

DATA STRUCTURES AND ITS APPLICATIONS

Multiple-Choice-Questions (MCQ's)

1. Which of the following best descri
nodes in an SLL?

a) Continuous memory blocks are used

b) Nodes are allocated dynamically and

c) Memory allocation is fixed during cor

d) Memory blocks are always allocated

**2. Which of the following is the main
doubly linked list (DLL)?**

a) SLL requires more memory per node

b) Traversal in reverse direction is not p

c) SLL insertion is slower.

d) SLL uses more dynamic memory allo

**2. Which of the following is the main
doubly linked list (DLL)?**

a) SLL requires more memory per node

b) Traversal in reverse direction is not p

c) SLL insertion is slower.

d) SLL uses more dynamic memory allo

3. In which scenario is it preferable

a) When random access is required f

b) When frequent insertions and delet

c) When memory must be allocated st

d) When reverse traversal is required.

DATA STRU

Vandana M L

Department of C

DATA STRUCTURES AND APPL

Doubly Linked List

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND APPLICATIONS

Doubly Linked List

Adoubly linked list contain three fields:

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

DATA STRUC

Vandana M L

Department of C

DATA STRUCTURES AND ITS A

Circular Singly Linked List

Vandana M L
Department of Computer Science and Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Circular Linked List

Circular linked list is a linked list where all no
connected to form a circle.

- Circular Singly Linked List
- Circular Doubly Linked List

  With additional head node
  Without additional head node

DATA STRUCTURES AND ITS APPLICATIONS

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

DATA STRUCTURES AND ITS APPLICATIONS

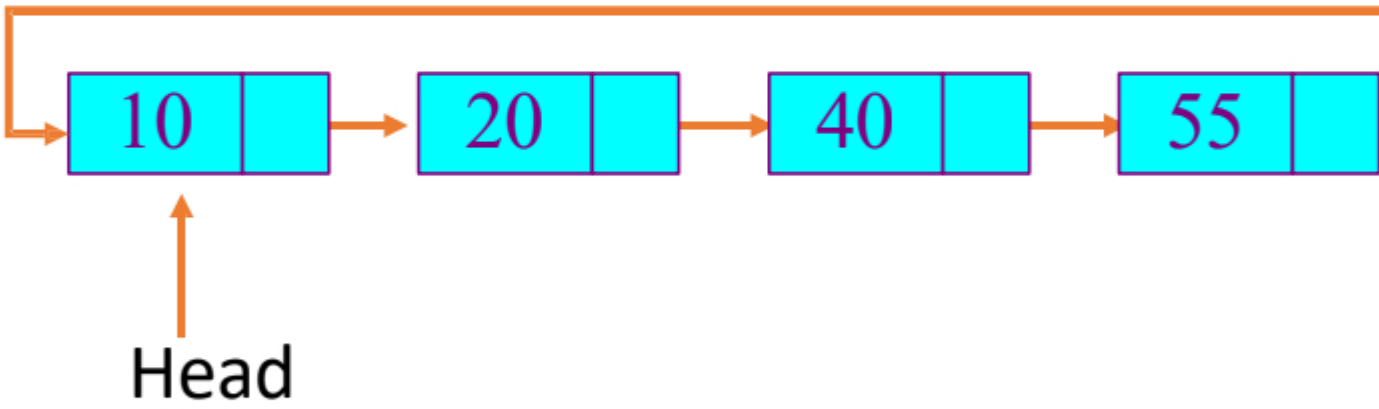
Circular Linked List: Applications

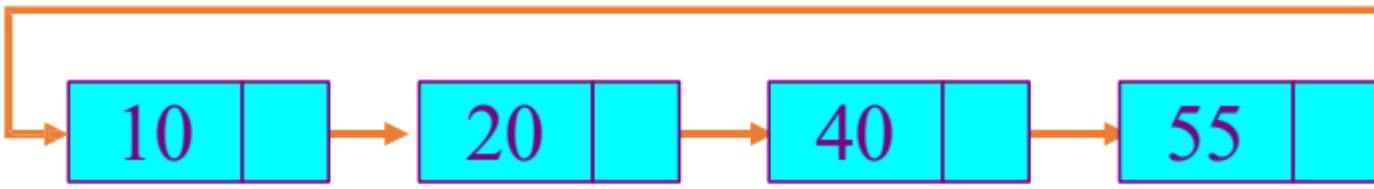
- Useful for implementation of queue, elim
  the need to maintain two pointers as in ca
  queue implementation using arrays
- Circular linked lists are useful for applicati
  repeatedly go around the list like playing v
  sound files in “looping” mode
- Advanced data structures like Fibonacci He
  Implementation
- Plays a key role in linked implementation o

DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List

- It supports traversing from the end of the beginning by making the last node point b the list
- A Tail pointer is often used instead of a He





DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List Node Definition

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

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

DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List Operations

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 nev

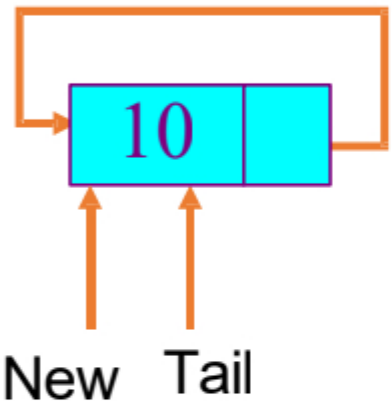
Else

- Change the new node link field to point to th
- Change the last node link to point to the nev

DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List Operations

Insertion into an empty list

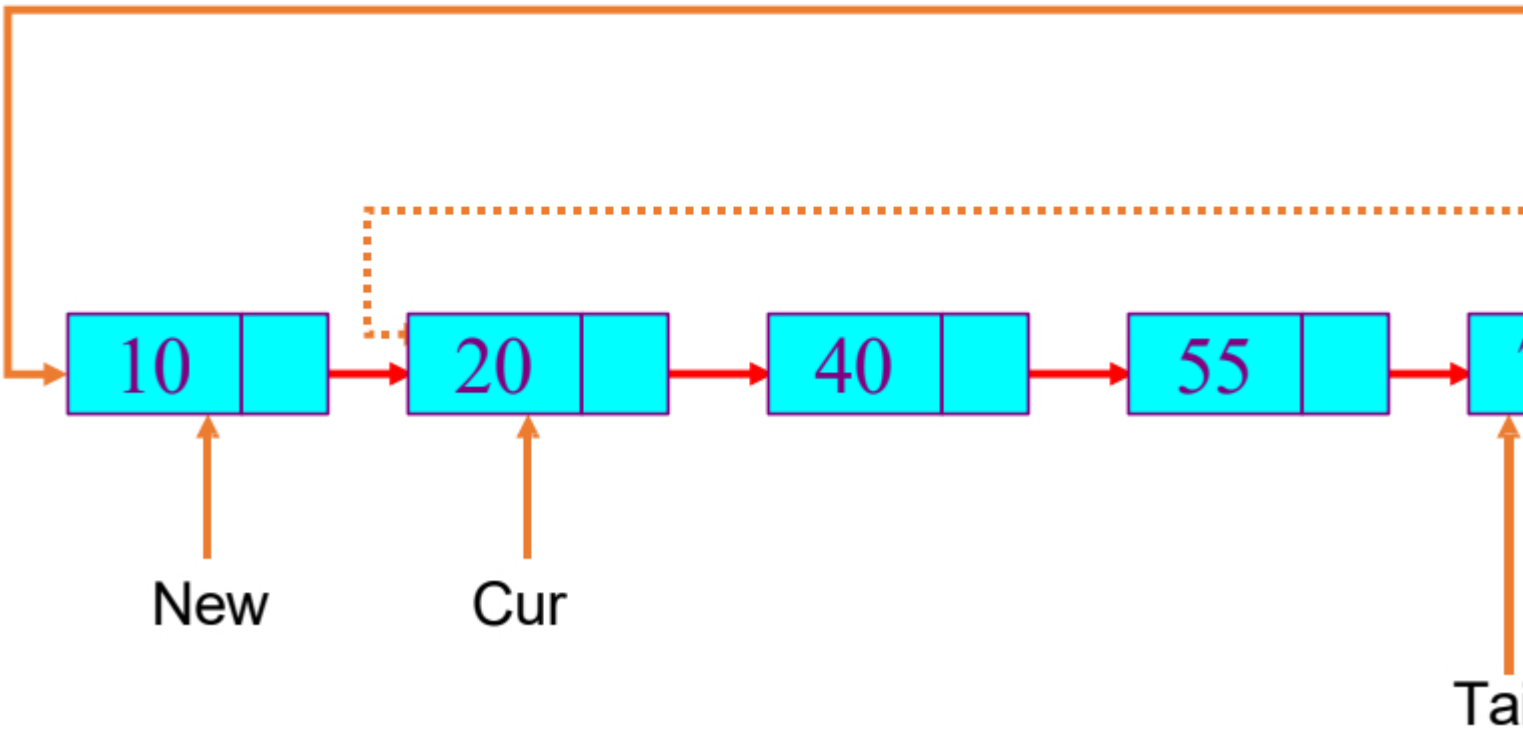


DATA STRUCTURES AND ITS APPLICATIONS
Circular Singly Linked List Operations

Insert to head of a Circular Linked List

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

Prev->next = New; ⟹ Tail->next = New;



DATA STRUCTURES AND ITS APPLICATIONS

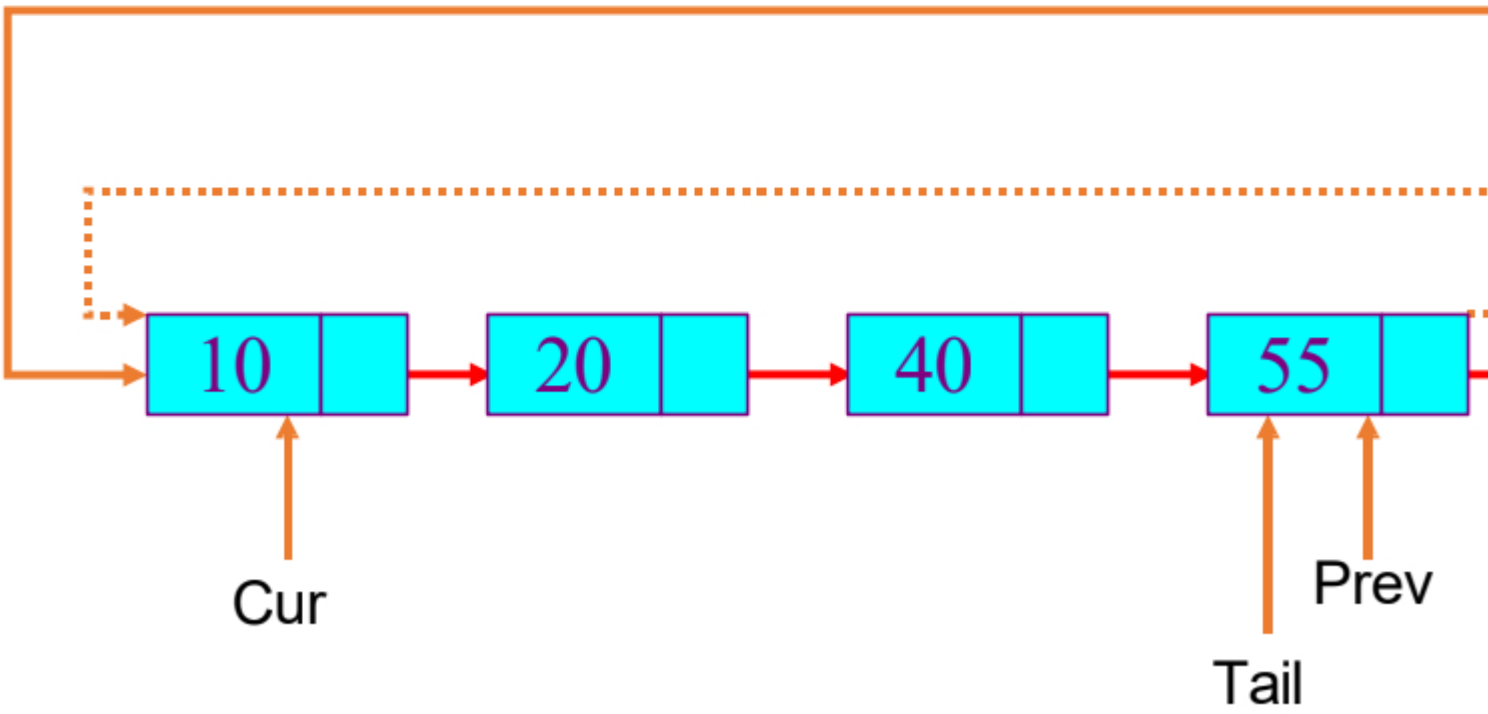
Circular Singly Linked List Operations

Insert to the end of a Circular Linked List

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

Prev->next = New; → Tail->next = New;

Tail = New;



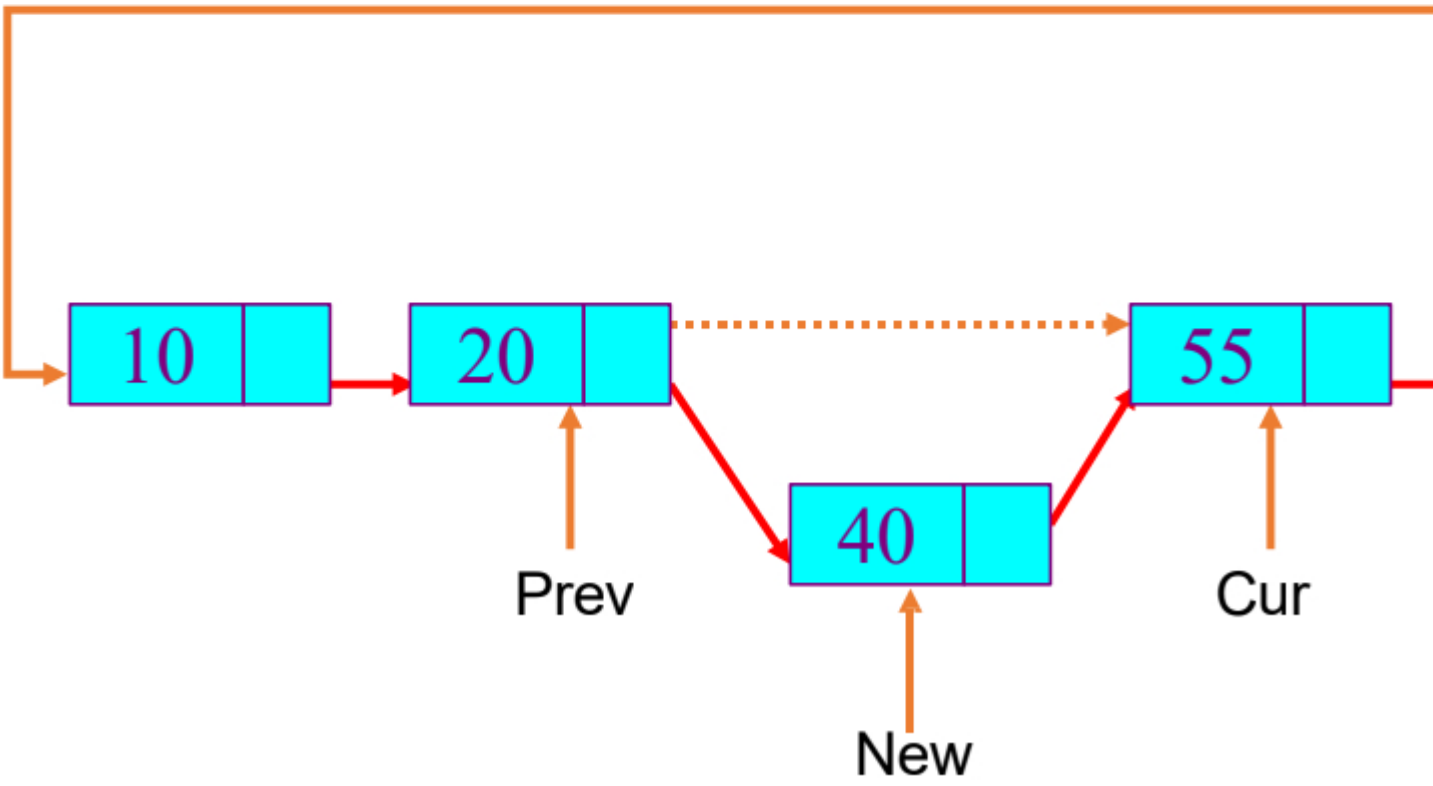
DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List Operations

Insert to the middle of Circular Linked List

New->next = Cur;

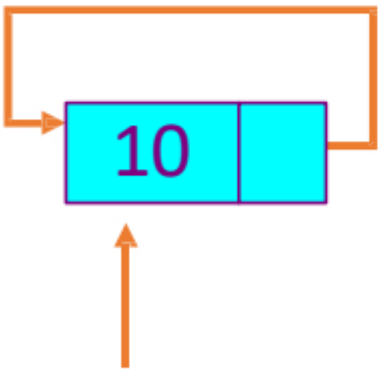
Prev->next = New;



Delete a node from a single-node Circular L

Tail = NULL;

free(Cur);



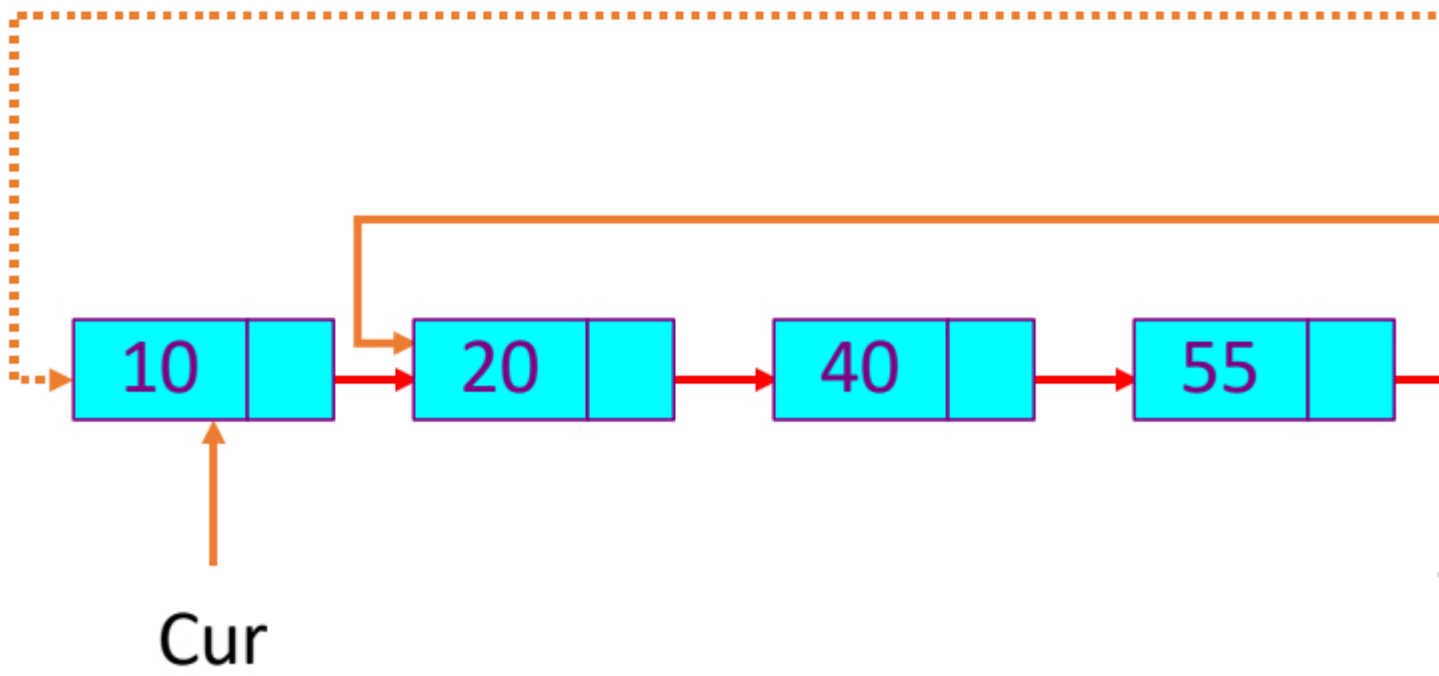
Tail = Cur = Prev

DATA STRUCTURES AND ITS APPLICATIONS

Circular Singly Linked List Operations

Delete the head node from a Circular Lin

```
Prev->next = Cur->next;          // same as: Tail
free(cur);
```

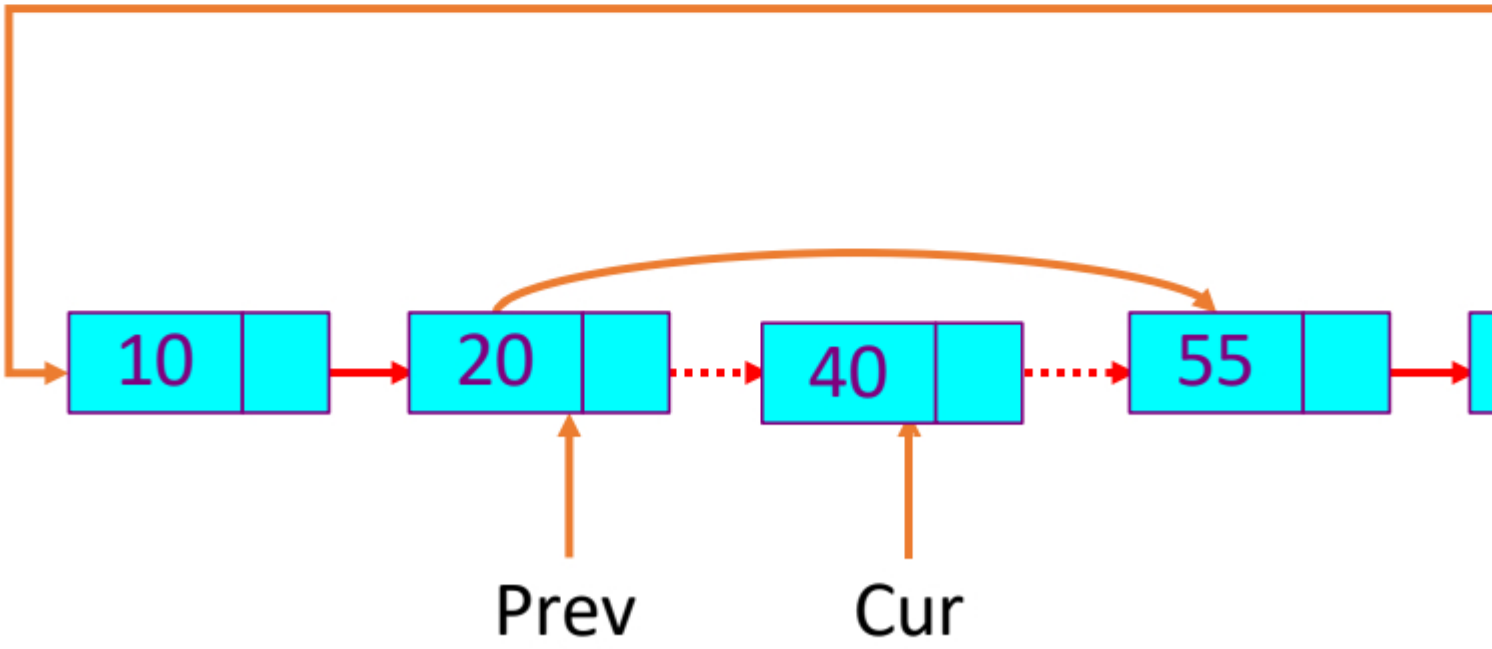


DATA STRUCTURES AND ITS APPLICATIONS
Circular Singly Linked List Operations

Delete a middle node Cur from a Circular

Prev->next = Cur->next;

Free(Cur);



DATA STRUCTURES AND ITS APPLICATIONS
Lecture Summary

Circular Singly Linked List operations

Apply the concepts to implement following o

- insert a node after a given node(point
- Insert a node after a node with a given

DATA STRUCTURES AND ITS APPLICATIONS
Multiple-Choice-Questions(MCQ's)

1. In a CSLL, which is the correct cond

a) while (temp != NULL)

b) while (temp->next != NULL)

c) do { ... } while (temp != head)

d) while (temp->next != head->next)

DATA STRUCTURES AND ITS APPLICATIONS
Multiple-Choice-Questions(MCQ's)

1. In a CSLL, which is the correct cond

a) while (temp != NULL)

b) while (temp->next != NULL)

c) do { ... } while (temp != head)

d) while (temp->next != head->next)

2. Which of the following is true about t

a) head always points to the last node.

b) head points to the first node, but last->n

c) head->next = NULL.

d) head cannot be NULL in any case.

2. Which of the following is true about t

a) head always points to the last node.

b) head points to the first node, but last->n

c) head->next = NULL.

d) head cannot be NULL in any case.

3. To insert a node at the beginning of following sequences is correct?

a) Add new node, set new->next = head, new, then update head = new.

b) Set new->next = head, update head =

c) Set new->next = head->next, update h

d) Set head->next = new, then new->nex

3. To insert a node at the beginning of following sequences is correct?

a) Add new node, set new->next = head, new, then update head = new.

b) Set new->next = head, update head =

c) Set new->next = head->next, update h

d) Set head->next = new, then new->nex

**4. For inserting a node at the end of a
mandatory?**

a) Update last->next = new and new->ne

b) Find last node (whose next = head), s
update last->next = new.

c) Set head = new if last node exists.

d) No need to link to head, as list is circu

4. For inserting a node at the end of a mandatory?

a) Update last->next = new and new->ne

b) Find last node (whose next = head), s update last->next = new.

c) Set head = new if last node exists.

d) No need to link to head, as list is circu

5. If a CSLL contains only one node ar

a) head = NULL.

b) head->next = head.

c) head->next = NULL.

d) Nothing, list remains unchanged.

5. If a CSLL contains only one node ar

a) head = NULL.

b) head->next = head.

c) head->next = NULL.

d) Nothing, list remains unchanged.

THANK Y

Vandana M L

Department o

vandanamd@

Data Str

Dinesh Sin

Departmer

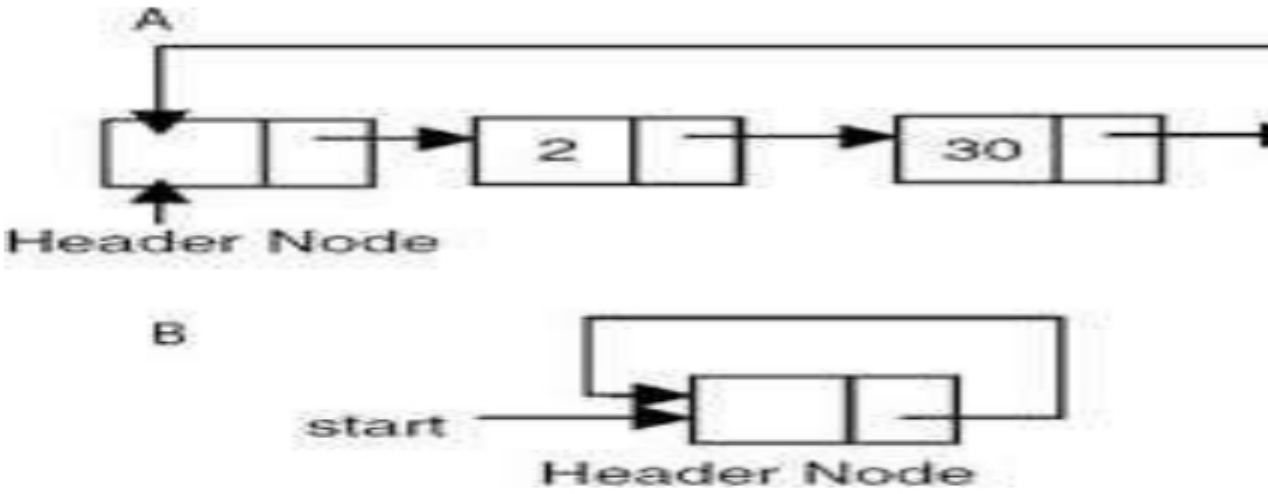
DATA STRUCTURES AND ITS

Circular Linked Lists

Dinesh Singh
Department of Computer Science & Engineering

Data Structures and its Applications

Circular Linked Lists



- The first node in the list is the header no
- The address part of the last node points t node
- Circular list does not have a natural first
- An External pointer points to the header one following this node is the first node.

Data Structures and its Applications

Operations on Circular Linked Lists

Implementation of some operations on Circula

- 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
the first node

Data Structures and its Applications

Operations on Circular Linked Lists

Creating Header node

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

Data Structures and its Applications

Operations on Circular Linked Lists

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

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

Operations on Circular Linked Lists with head

```
void insert_head(struct node *p,int x)
//p points to the header node, x element to be i
 {
  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
   temp->next=p->next;
   p->next=temp; //next part of header node poi
   p->data++;
  }
```

Data Structures and its Applications

Operations on Circular Linked Lists with hea

Algorithm to insert a node at the end of the

insert_tail(p,x)

//p pointer to header node, x element to b

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 n

Copy the address of the new node into the

Data Structures and its Applications

Operations on Circular Linked Lists with hea

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
 q->next=temp; // copy the address of new node int
 p->data++; // increment the count of nodes in the li
 }
```

Data Structures and its Applications

Operations on Circular Linked Lists with hea

Algorithm to delete a node given its value

```
delete_node(p,x)
//p pointer to header node, x element to b

move forward until the node to be deleted
or header node is reaches
If(node found)
   delete the node by adjusting the pointers
else
  node not found // if header node is reache
```

Operations on Circular Linked Lists with hea

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

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

Operations on Circular Linked Lists with hea

```
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 i
  }
}
```

Data Structures and its Applications

Multiple-Choice-Questions (MCQ's)

1. **In a circular singly linked list (CSL)**
   primary advantage of the header no

a) It stores data just like other nodes.

b) It acts as a sentinel to simplify insertion

c) It prevents memory leaks.

d) It is used only for maintaining a count

Data Structures and its Applications

Multiple-Choice-Questions (MCQ's)

1. **In a circular singly linked list (CSL)**
   primary advantage of the header n

a) It stores data just like other nodes.

b) It acts as a sentinel to simplify insertio

c) It prevents memory leaks.

d) It is used only for maintaining a count

2. To traverse a CSL with a header n
conditions is most appropriate?

a) while (ptr != NULL)

b) while (ptr != header) (starting from h

c) while (ptr->next != NULL)

d) while (ptr->next != header->next)

2. To traverse a CSL with a header n
conditions is most appropriate?

a) while (ptr != NULL)

b) while (ptr != header) (starting from h

c) while (ptr->next != NULL)

d) while (ptr->next != header->next)

**3. When inserting a node immediately
minimum number of pointer updates r
node?**

a) 1

b) 2

c) 3

d) It depends on list size.

3. When inserting a node immediately
minimum number of pointer updates r
node?

a) 1

b) 2

c) 3

d) It depends on list size.

4. When deleting the last data node (ju
with a header node, which of the follov

a) Find the second-last node, make its ne
node.

b) Free header node and reconnect list.

c) Move header pointer one step backwa

d) Remove the first node instead of the la

**4. When deleting the last data node (ju
with a header node, which of the follov**

a) Find the second-last node, make its ne
node.

b) Free header node and reconnect list.

c) Move header pointer one step backwa

d) Remove the first node instead of the la

5. When searching for a key in CSL witl
store data), which is the correct loop te

a) while (ptr != header)

b) while (ptr != NULL)

c) while (ptr->next != NULL)

d) while (ptr->next != header)

5. When searching for a key in CSL witl
store data), which is the correct loop te

a) while (ptr != header)

b) while (ptr != NULL)

c) while (ptr->next != NULL)

d) while (ptr->next != header)

TH

Dine

Dep