K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Data Structure

ENCS205

School of Engineering & Technology (SOET)
K.R. Mangalam University

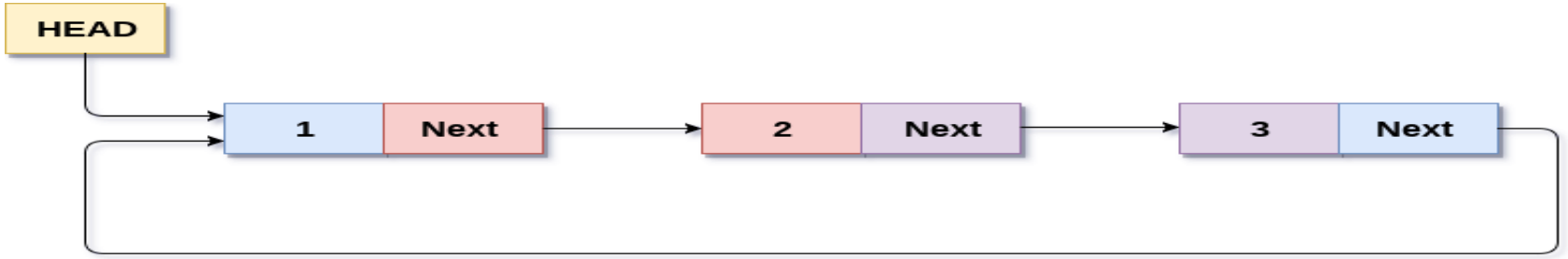
UNIT-2

Session 20: Circular linked list

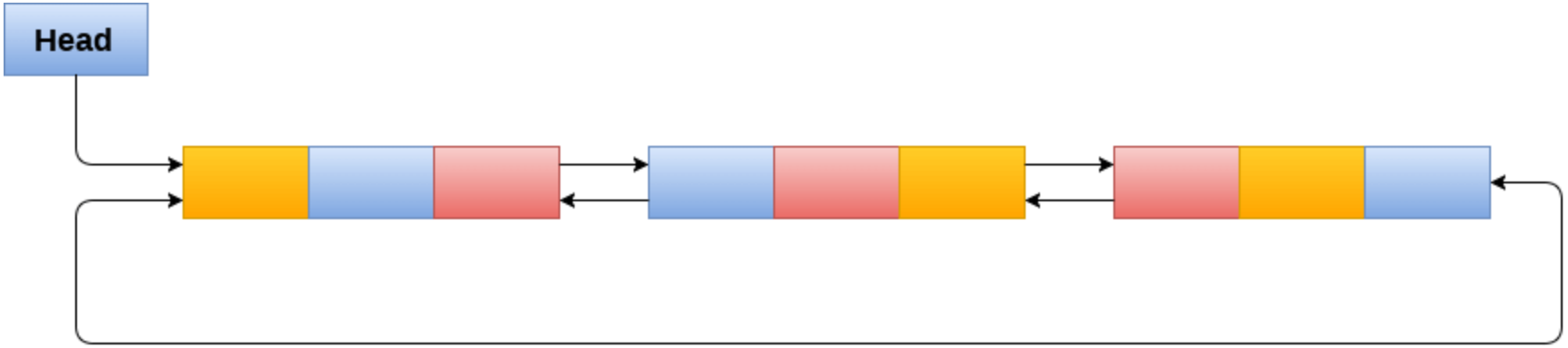
Sessions 20 Outlook

- Circular linked list
- Memory Representation
- Application of Circular Linked List
- Operations
- Header Linked List
- Applications
- Assignment

Circular linked list

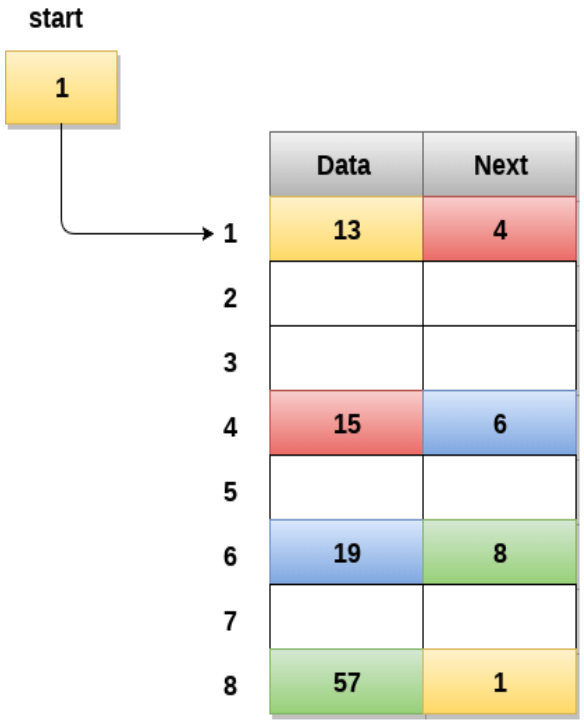


Circular Singly Linked List

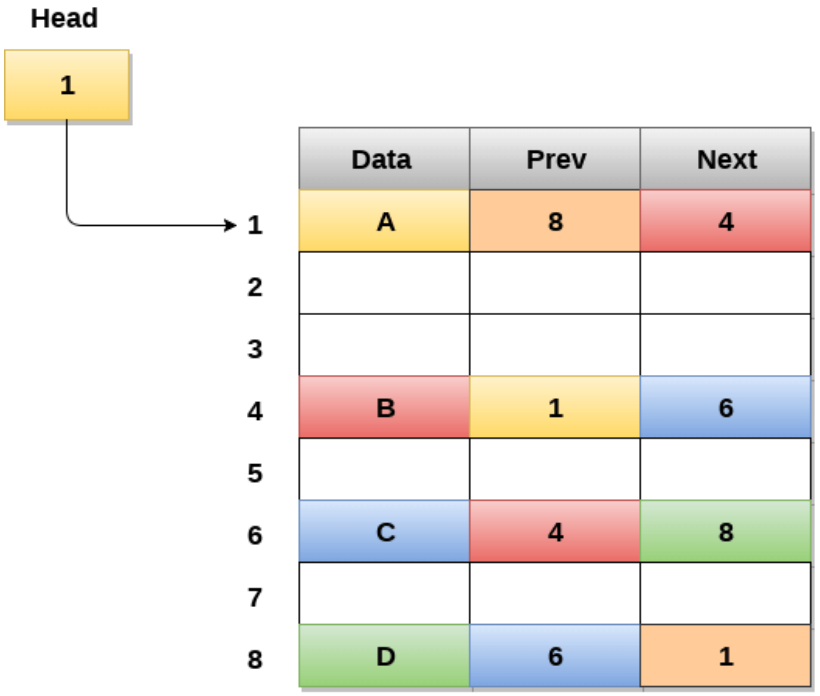


Circular Doubly Linked List

Memory Representation



Memory Representation of a circular linked list



Memory Representation of a Circular Doubly linked list

Circular linked list

Advantages of a Circular linked list

- Entire list can be traversed from any node.
- Circular lists are the required data structure when we want a list to be accessed in a circle or loop.
- Despite of being singly circular linked list we can easily traverse to its previous node, which is not possible in singly linked list.

Disadvantages of Circular linked list

- Circular list are complex as compared to singly linked lists.
- Reversing of circular list is a complex as compared to singly or doubly lists.
- If not traversed carefully, then we could end up in an infinite loop.
- Like singly and doubly lists circular linked lists also doesn't supports direct accessing of elements

Application of Circular Linked List

- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.

- Another example can be Multiplayer games. All the Players are kept in a Circular Linked List and the pointer keeps on moving forward as a player's chance ends.

- Circular Linked List can also be used to create Circular Queue. In a Queue we have to keep two pointers, FRONT and REAR in memory all the time, where as in Circular Linked List, only one pointer is required.

Operations

- Creation of list
- Traversal of list
- Insertion of node
  - At the beginning of list
  - At any position in the list
- Deletion of node
  - Deletion of first node
  - Deletion of node from middle of the list
  - Deletion of last node
- Counting total number of nodes
- Reversing of list

Code for Insertion

```
void insert(int newdata) {
   struct Node *newnode = (struct Node *)malloc(sizeof(struct Node));
   struct Node *ptr = head;
   newnode->data = newdata;
   newnode->next = head;
   if (head!= NULL) {
      while (ptr->next != head)
      ptr = ptr->next;
      ptr->next = newnode;
   } else
   newnode->next = newnode;
   head = newnode;
}
```

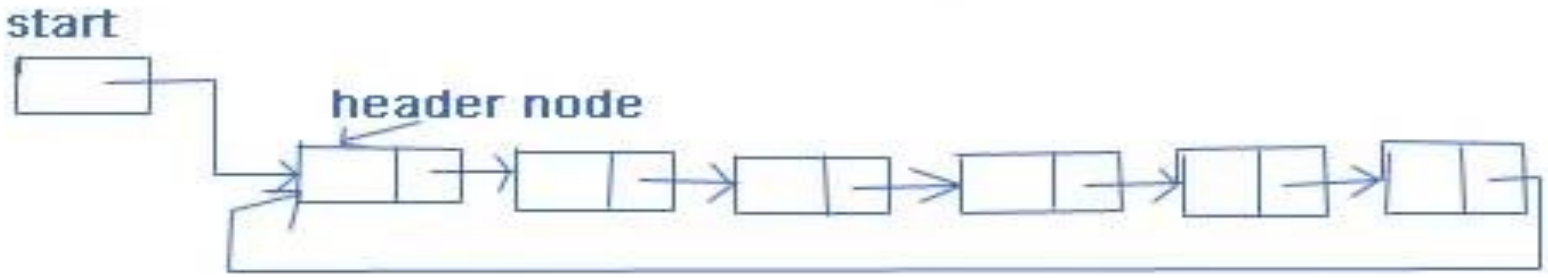
Code for Display

```
void display() {
   struct Node* ptr;
   ptr = head;
   do {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
   } while(ptr != head);
}
```

Complete Code here

Header Linked List

(a) Grounded header list.



(b) Circular header list.

Header Linked List

- A header list is a linked list, which always contains a special node, called header node, at the beginning of the linked list.
- This header node usually contains vital information about the linked list such as the number of nodes in the list, whether the list is sorted or not.

Structure of the list

```
/ Structure of the list
struct link {
    int info;

    // Pointer to the next node
    struct link* next;
};
```

create header linked list

```
struct link* create_header_list(int data)
{

    // Create a new node
    struct link *new_node, *node;
    new_node = (struct link*)malloc(sizeof(struct link));
    new_node->info = data;
    new_node->next = NULL;
```

Complete code here

Review unit 2

- Stack &its Operations
- Queue & its type
- Linked list & operations
- Applications

Stack: Definition, Operations, Applications

Definition

1

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle, where the last element added is the first one to be removed.

Operations

2

The core stack operations are push (add an element), pop (remove the top element), and peek (view the top element).

Applications

3

Stacks are widely used in programming, programming, such as for expression expression evaluation, function calls, and and backtracking algorithms.

Queue: Definition, Operations, Applications

1 Definition

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle, where the first element added is the first one one to be removed.

2 Operations

The main queue operations are enqueue (add an element to the rear), rear), dequeue (remove an element element from the front), and peek (view the front element).

3 Applications

Queues are used in real-world world scenarios like task scheduling, job processing, processing, and event handling, where the order of of elements is crucial.

Linked List: Definition, Types, Operations

Definition

A linked list is a linear linear data structure structure where each each element (called (called a node) contains data and a a reference (or link) to to the next node in the list.

Types

There are several types of linked lists, including singly, doubly, and circular linked lists, each with its own advantages and use cases.

Operations

The key linked list operations are insertion, deletion, and traversal, which which can be performed efficiently efficiently due to the the dynamic nature of of the data structure. structure.

Practical Use Cases for Data Structures

Undo/Redo Operations

Stacks are used to to implement undo and redo functionality in software applications.

Print Spooling

Queues are used used to manage print jobs in a printer spooler, ensuring a fair and and orderly processing of requests.

Breadth-First Search

Queues are used used in graph traversal algorithms, such such as Breadth-Breadth-First Search (BFS), to explore nodes in a in a specific order. order.

Memory Management

Linked lists are used for dynamic dynamic memory memory allocation allocation and deallocation in operating systems systems and runtime environments.

Stack Vs Queues

Stack

- Stacks are based on the LIFO principle.
- Insertion and deletion in stacks takes place only from one end of the list called the top.
- Operation: Push, Pop

Queue

- Queues are based on the FIFO principle.
- Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
- Operation: Enqueue, Dequeue.

Arrays vs. Linked Lists

Array

- Stored in Contiguous Locations
- Fixed in Size
- Uses less Memory.
- Elements can be accessed easily

LinkedList

- Stored in non-Contiguous locations.
- Dynamic in Size.
- Uses more memory because it stores both data and node address
- Elements are accessed by traversing the whole Linked-List.

Exercise

1. If a stack is initially empty and you perform 5 push operations followed by 3 pop operations, what will be the size of the stack?

2. Given a stack with a maximum capacity of 10 elements, what will happen if you try to push an element onto a full stack?

3. If a stack contains 10 elements and you perform 4 push operations and 6 pop operations, what will be the current size of the stack?

4. Assume a stack implementation using an array with a maximum capacity of 5 elements. If you push elements 2, 4, 6, and 8 onto the stack, what will be the top element of the stack?

5. Suppose you have a stack with elements [3, 6, 9, 12]. If you perform a pop operation followed by a push operation with the value 15, what will be the top element of the stack?

6. Implement a stack using an array with a capacity of 5 elements. Perform push operations with values 2, 4, and 6. What will be the resulting array?

7. Given an array-based stack with elements [10, 20, 30, 40, 50], perform two pop operations. What will be the resulting stack?

8. If you push elements onto an array-based stack until the array is full and then perform a pop operation, what will be the result?

9. Implement a stack using an array with a capacity of 6 elements. Perform push operations with values 1, 3, 5, 7, and What will be the top element of the stack?

11.Convert the infix expression "4 * (3 + 7)" to postfix notation.

12.Convert the infix expression "(8 - 2) / (5 + 3)" to postfix notation.

13.Convert the postfix expression "5 3 * 8 +" to infix notation.

14.Convert the postfix expression "7 2 + 5 * 3 /" to infix notation.

15.Convert the prefix expression "* + 3 5 - 2 4" to infix notation.

16.How many moves are required to solve the Tower of Hanoi problem with 3 disks?

17.If you have 5 disks in the Tower of Hanoi problem, how many moves are needed to solve it?

18.If you pop an element from an empty array-based stack, what will be the result?

17. If a queue is initially empty and you enqueue 5 elements followed by dequeueing 3 elements, what will be the size of the queue?
18. Given a queue with a maximum capacity of 10 elements, what will happen if you try to enqueue an element onto a full queue?
19. If a queue contains 10 elements and you perform 4 enqueue operations and 6 dequeue operations, what will be the current size of the queue?
20. Implement a queue using an array with a capacity of 5 elements. Perform enqueue operations with values 2, 4, 6, and 8. What will be the resulting array?
21. If you dequeue an element from an empty array-based queue, what will be the result?

23.Given an array-based linear queue with elements [10, 20, 30, 40, 50], perform two dequeue operations. What will be the resulting queue?

24.If you enqueue elements onto an array-based linear queue until the array is full and then perform a dequeue operation, what will be the result?

25.Given a linear queue implemented using an array with a capacity of 6 elements, if the front and rear indices are both at position 2, what will be the current size of the queue?

26.Implement a linear queue using an array with a capacity of 4 elements. Perform enqueue operations with values 2, 4, 6, and 8. What will be the front and rear indices of the queue after these operations?

27.Implement a circular queue using an array with a capacity of 5 elements. Perform enqueue operations with values 2, 4, 6, and 8. What will be the front and rear indices of the queue after these operations?

28.Given an array-based circular queue with elements [10, 20, 30, 40, 50], perform two dequeue operations. What will be the resulting queue?

29.If you enqueue elements onto an array-based circular queue until the array is full and then perform a dequeue operation, what will be the result?

30.Given a circular queue implemented using an array with a capacity of 6 elements, if the front and rear indices are both at position 5, what will be the current size of the queue?

31.Implement a circular queue using an array with a capacity of 4 elements. Perform enqueue operations with values 2, 4, 6, and 8. What will be the front and rear indices of the queue after these operations?

32.Implement a priority queue using an array-based implementation. Perform enqueue operations with values 10, 20, 30, and 15 with priorities 2, 1, 3, and 2 respectively. What will be the resulting priority queue?

33.Given a priority queue with elements [30, 20, 40, 10] and corresponding priorities [3, 2, 4, 1], perform two dequeue operations. What will be the resulting priority queue?

34.If you enqueue elements with priorities into a priority queue until it is full and then perform a dequeue operation, what will be the result?

35.Implement a priority queue using an array with a capacity of 5 elements. Perform enqueue operations with values 8, 5, 10, and 7 with priorities 1, 2, 1, and 3 respectively. What will be the front and rear indices of the priority queue after these operations?


THANK
YOU