LINKED LIST DATA STRUCTURE

Content

3.1. Print Middle Element

1. Using Counting : O(n)

```
int middleEle(Node head)
{
        int len = 0;
        Node temp = head;
        while(temp!=null){
                len++;
                temp = temp.next;
        }
        temp = head;
        for(int i=0;i<len/2;i++){
                temp = temp.next;
        }
        return temp.data;
}
```

2. Using Slow and Fast Pointer : O(n)

```
int printMiddle()
{
        Node slow_ptr = head;
        Node fast_ptr = head;
        if (head != null){
                while (fast_ptr != null && fast_ptr.next != null) {
                        fast_ptr = fast_ptr.next.next;
                        slow_ptr = slow_ptr.next;
                }
        }
        return slow_ptr.data;
}
```

3.2. Nth node from end of linked list

```
int getNthFromLast ( Node head , int n )
{
        ArrayList<Integer> list = new ArrayList<>();
        Node temp = head;
        int len = 0;
        while(temp!=null){
           len++;
           list.add(temp.data);
           temp = temp.next;
        }
        if ( len > = n){
           return list.get ( len – n );
        }
        return -1;
}
```

3.3. Rotate a Linked List

```
public Node rotate ( Node head , int k) {
        int len = 1;
        Node temp = head;
        while ( temp.next != null ) {
           len++;
           temp = temp.next;
        }
        int newHead = k % len;
        if( newHead > 0) {
           temp.next = head;
           temp = head;
           for ( int i = 0 ; i < newHead – 1 ; i++){
              temp = temp.next;
           }
           head = temp.next;
           temp.next = null;
        }
        return head;
}
```

Input: 2 4 7 8 9 k : 3

output: 8 9 2 4 7

3.4. Reverse a Linked List in groups of given size

Reverse List

```
Node reverseList ( Node head )
{
      Node pre = null , curr = head , nex = null;
      while ( curr != null ){
         nex = curr.next;
         curr.next = pre;
         pre = curr;
         curr = nex;
      }
      head = pre;
      return head;
}
```

Reverse in group of given size

```
public static Node reverse(Node node, int k){
      int flag = 0;
      Node pre, curr = node , nex = null, cu = node;
      while (curr != null){
         Node start = curr;
         int i=0;
         pre = null;
         while (curr != null && i<k){
            nex = curr.next;
            curr.next = pre;
            pre = curr;
            curr = nex;
            i++;
         }
         if(flag == 0){
            flag = 1;
            node = pre;
         }else{
            cu.next = pre;
            cu = start;
         }
      }
      return node;
   }
```

3.5. Check if Linked List is Palindrome

```
boolean isPalindrome(Node head)
{
      ArrayList<Integer> list = new ArrayList();
      Node temp = head;
      while ( temp != null ){
         list.add(temp.data);
         temp = temp.next;
      }
      temp = head;
      int s = list.size() -1;
      while(temp != null){
         int d = list.get(s--);
         if(d != temp.data)
            return false;
         temp = temp.next;
      }
      return true;
}
```

3.6. Detect Loop in linked list

```
int detectLoop ( Node head )
{
      Node slow = head;
      Node fast = head;
      while ( fast != null && fast.next != null ) {
         slow = slow.next;
         fast = fast.next.next;

         if(slow == fast){
            return 1;            // loop
         }
      }
      return 0;            //no loop
}
```

3.7. Remove loop in Linked List

```
public static void removeLoop ( Node head )
{
      Node slow = head , fast = head;
      while ( fast != null && fast.next != null){
         slow = slow.next;
         fast = fast.next.next;

         if ( slow == fast){   // Loop found
            Node ptr1 = head , ptr2 = null;
            while( true ){
                ptr2 = slow;
                while ( ptr2.next != slow && ptr2.next != ptr1){
                    ptr2 = ptr2.next;
                }

                if(ptr2.next == ptr1){
                    break;
                }

                ptr1 = ptr1.next;
            }
            ptr2.next = null;
         }
      }
}
```

3.10. Given a linked list of 0s, 1s and 2s, sort it : O(n)

```
class LinkedList
{
   static Node segregate(Node head)
   {
      Node temp = head;
      int arr[] = new int[3];
      while(temp!=null){  //count no. of occurrence
         int d = temp.data;
         arr[d]++;
         temp = temp.next;
      }
      temp = head;
      for ( int i = 0 ; i < 3 ; i++){
         int size = arr[i];
         int j = 0;
         while ( j < size ) {
            temp.data = i ;
            temp = temp.next;
            j++;
         }
      }
      return head;
   }
}
```

3.11. Pairwise swap of nodes in LinkedList : O(n)

```
class Swap
{
   public static Node pairwise_swap ( Node node)
   {
      Node temp = node;
      while( temp !=null && temp.next!=null){
         Node nex = temp.next;
         int d = temp.data;
         temp.data = nex.data;
         nex.data = d;
         temp = nex.next;
      }
      return node;
   }

}
```

3.12. Add two numbers represented by linked lists : O(m+n)

```
Node addTwoLists ( Node first , Node second)
{
      int sum = 0 , carry = 0;
      Node res = null , prev = null;
      while ( first !=null || second != null ) {
          sum = carry;
          if ( first != null ){
              sum += first.data;
              first = first.next;
          }
          if ( second != null){
              sum += second.data;
              second = second.next;
          }

          carry = (sum >= 10) ? 1 : 0 ;
          sum = sum % 10;

          Node temp = new Node(sum);

          if (res == null)
              res = temp;
          else
             prev.next = temp;

          prev = temp;
          sum = 0;
      }
      if ( carry > 0){   //last
          prev.next = new Node (carry);
      }
      return res;
}
```

3.13. Implement Queue using Linked List

```
void push (int a)
{
        QueueNode node = new QueueNode(a);
        if(rear == null){
            rear = node;
            front = rear;
        }else{
             rear.next = node;
             rear = rear.next;
         }
}

int pop()
{
      if(front != null){
           int d = front.data;
           if ( front == rear){
                front = rear = null;
           }else{
                front = front.next;
           }
           return d;
      }
      return -1;
}
```

2.14. Implement Stack using Linked List

```
StackNode top = null;
void push(int a)
{
        StackNode node = new StackNode(a);
        node.next = top;
        top = node;
}


int pop( ) {
        if (top != null){
           int d = top.data;
           top = top.next;
           return d;
        }
        return -1;
}
```

2.15. Delete without head pointer

```
class GfG
{
   void deleteNode ( Node node )
   {
        Node nextNode = node.next;
        //swap value of node and its next node
        int temp = node.data;
        node.data = nextNode.data;
        nextNode.data = temp;

        //delete nextNode
        node.next = nextNode.next;
   }
}
```

2.16. Flattening a linked list

```
class GfG
{
        Node merge( Node a, Node b){
                Node result = null;
                //base case
                if(a==null)
                        return b;
                else if(b == null)
                        return a;

                if(a.data <= b.data){
                        result = a;
                        result.next = merge( a.bottom, b);
                }else{
                        result = b;
                        result.next = merge( a, b.bottom);
                }
                return result;
        }
        Node flatten(Node root)
        {
                Node temp = root, m = root;
                while( temp.next != null){
                        temp = temp.next;
                        m = merge(m, temp);
                }
                return m;
        }
```