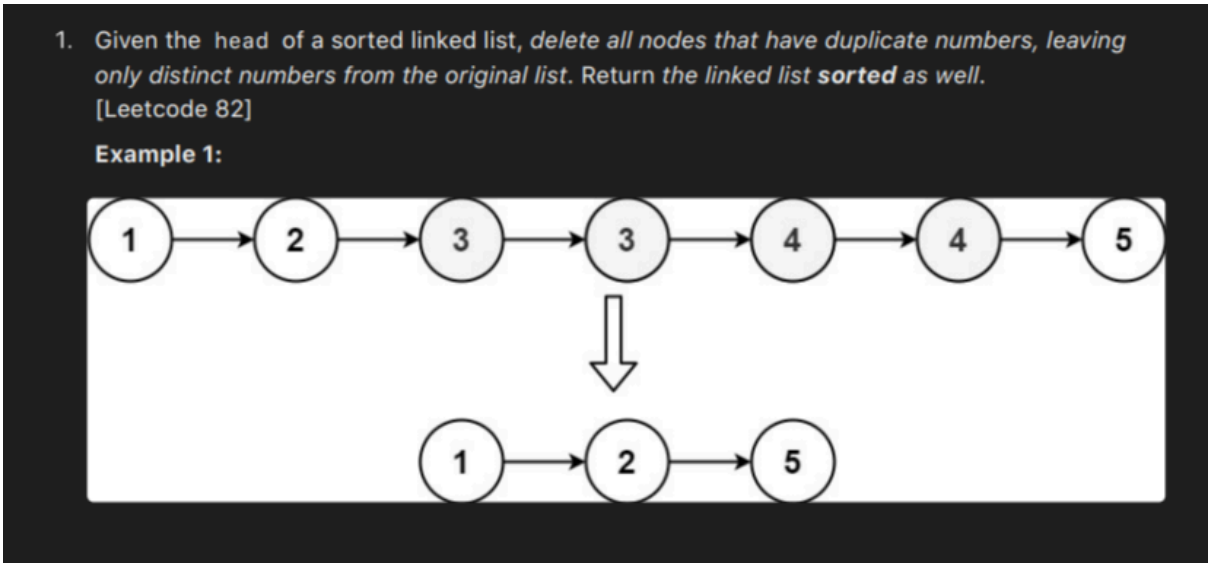
NAME:SAURABH RAJ
REGISTERED EMAIL :saurabhraj25aug2004@gmail.com
COURSE NAME:DECODE DSA WITH C++
BATCH:DECODE 2.0
MODULE NAME:LINKED LIST Assignment part 3
MOBILE NUMBER:8434283953

—----------

QUESTION1:



Answer:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode *dummy = new ListNode(0);
        dummy->next = head;
        ListNode *temp = dummy;
        while(head != NULL){
            if(head->next and head->val == head->next->val){
                while(head->next and head->val == head->next->val){
                    head = head->next;
                }
                temp->next = head->next;
            }
            else temp = temp->next;
                head = head->next;
        }
        return dummy->next;
    }
};
```

Question:2

2. Given the `head` of a singly linked list, sort the list using **insertion sort**, and return *the sorted list's head*. [Leetcode 147]

   The steps of the **insertion sort** algorithm:

   1. Insertion sort iterates, consuming one input element each repetition and growing a sorted output list.
   2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there.
   3. It repeats until no input elements remain.

   The following is a graphical example of the insertion sort algorithm. The partially sorted list (black) initially contains only the first element in the list. One element (red) is removed from the input data and inserted in-place into the sorted list with each iteration.

Answer:

```
class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {
        ListNode *dummy = new ListNode(0);
        ListNode *temp = dummy;
        ListNode *curr = head , *nex = NULL;
        while(curr){
            nex = curr->next;
            temp = dummy;
            while(temp->next and temp->next->val < curr->val)
                temp = temp->next;
            curr->next = temp->next;
            temp->next = curr;
            curr = nex;
        }
        return dummy->next;
    }
};
```

Question:3

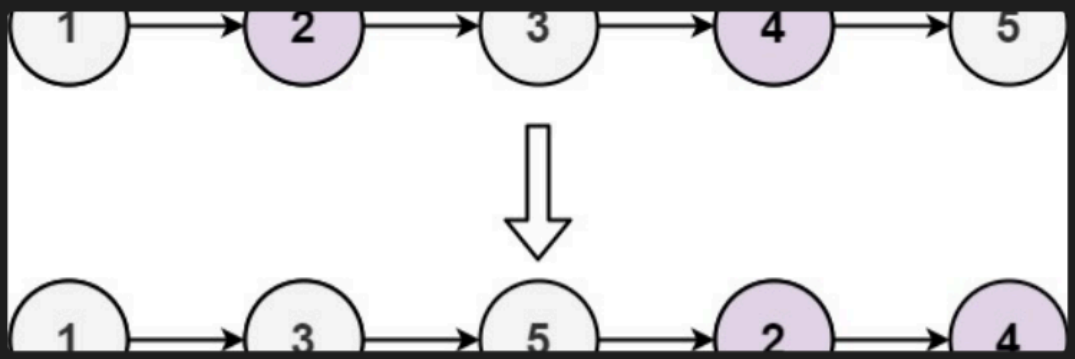
3. Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*. [Leetcode 328]

   The **first** node is considered **odd**, and the **second** node is **even**, and so on.

   Note that the relative order inside both the even and odd groups should remain as it was in the input.

   You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

   Example 1:



Answer:

```cpp
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(!head or !head->next)return head;

        ListNode *odd=head;
        ListNode *eve=head->next;
        ListNode *newhead=eve;
        while(eve and eve->next){
            odd->next=eve->next;
            odd=odd->next;
            eve->next=odd->next;
            eve=eve->next;
        }
        odd->next=newhead;
        return head;
    }
};
```

Question:4

4. You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. [Leetcode 2]

   You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Answer:

```cpp
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
       int carry = 0;
        if(!l1)return l2;
        if(!l2)return l1;

        ListNode *a = l1;
        ListNode *b = l2;
        ListNode *c = new ListNode (0);
        ListNode *head = c;
        while(a or b){
            int x = a ? a->val : 0;
            int y = b ? b->val : 0;
            int sum = x + y + carry;
            head->next = new ListNode(sum%10);
            carry = sum/10;
            head = head->next;
            if(a)a = a->next;
            if(b)b = b->next;
        }
        if(carry)head->next = new ListNode(carry);
        return c->next;
```

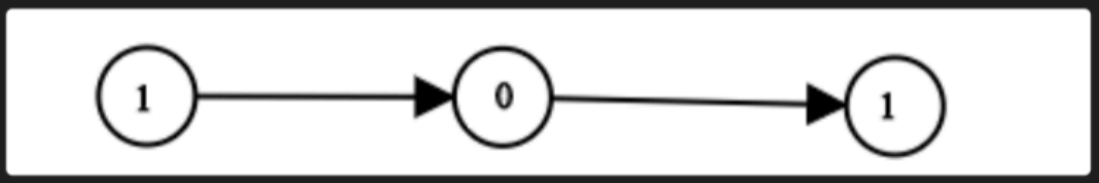
Question:5

5. Given `head` which is a reference node to a singly-linked list. The value of each node in the linked list is either `0` or `1`. The linked list holds the binary representation of a number.

   Return the *decimal value* of the number in the linked list. [Leetcode 1290]

   The **most significant bit** is at the head of the linked list.

   Example 1:



Answer:

```cpp
class Solution {
public:
    ListNode* reverse(ListNode *&head){
        ListNode *nex;
        ListNode *curr = head , *prev = NULL;
        while(curr){
           nex = curr->next;
           curr->next = prev;
           prev = curr;
            curr = nex;
        }
        head = prev;
        return head;
    }
    int getDecimalValue(ListNode* head) {
        head = reverse(head);
        int val = 1;
        int ans = 0;
        ListNode *tmp = head;
        while(tmp){
            ans+=val*tmp->val;
            tmp=tmp->next;
            val<<=1;
```

Question:6

6. In a linked list of size `n`, where `n` is **even**, the `ith` node (**0-indexed**) of the linked list is known as the **twin** of the `(n-1-i)th` node, if `0 <= i <= (n / 2) - 1`. [Leetcode 2130]

- For example, if `n = 4`, then node `0` is the twin of node `3`, and node `1` is the twin of node `2`. These are the only nodes with twins for `n = 4`.

The **twin sum** is defined as the sum of a node and its twin.

Given the `head` of a linked list with even length, return *the **maximum twin sum** of the linked list.*

Example 1:

Answer:

```
Solution :
class Solution {
```

```
public:
ListNode *middle(ListNode *head){
ListNode *fast = head->next;
ListNode *slow = head;
while(fast and fast->next){
fast = fast->next->next;
slow = slow->next;
}
return slow;
}
ListNode* reverse(ListNode *mid){
ListNode *curr = mid;
ListNode *prev = NULL;
while(curr){
ListNode *plus = curr->next;
curr->next = prev;
prev = curr;
curr = plus;
}
return prev;
}
int pairSum(ListNode* head) {
ListNode *mid = middle(head);
int sum = 0;
int maxi = 0;
ListNode *p = reverse(mid);
mid->next = NULL;
ListNode *temp = head;
while(temp and p){
sum = p->val + temp->val;
maxi = max(maxi , sum);
p = p->next;
temp=temp->next;
10
}
return maxi
```

Question:7

7. Given the `head` of a linked list, reverse the nodes of the list `k` at a time, and return *the modified list*. [Leetcode 25]

Answer:

```
class Solution {
public:
int length(ListNode *head){
int len = 0;
while(head){
head=head->next;
len++;
}
return len;
}
ListNode* reverseKGroup(ListNode* head, int k) {
int len = length(head);
if(!head or len<k)return head;
ListNode *dummy = new ListNode(0);
dummy->next = head;
ListNode* curr = dummy;
ListNode* prev = dummy;
ListNode* nex = dummy;
while(len>=k){
curr = prev->next;
nex = curr->next;
for(int i=1;i<k;i++){
curr->next = nex->next;
nex->next = prev->next;
prev->next = nex;
nex = curr->next;
}
prev = curr;
len-=k;
}
return dummy->next;
}
}
```