GeeksMan
Linked List
Lesson 3

Linked List of strings forms a palindrome

Given a linked list of strings having n nodes check to see whether the combined string formed is palindrome or not.

Example:

Input :

5

a bc d dcb a

4

a bc d ba

Output :

True

False

```
1.  bool compute(Node* root)
2.  {
3.     string s="";
4.    while(root!=NULL)
5.    {
6.       s+=root->data;
7.       root=root->next;
8.    }
9.    int n=s.length();
10.   for(int i=0;i<(n-1)/2;i++)
11.   {
12.      if(s[i]!=s[n-1-i])
13.      return false;
14.   }
15.   return true;
16. }
```

Intersection of two sorted Linked lists

Given two lists sorted in increasing order, create a new list representing the intersection of the two lists. The new list should be made with its own memory — the original lists should not be changed.

Example :

Input:

L1 = 1->2->3->4->6

L2 = 2->4->6->8

Output: 2 4 6

Approaches:

1)Traverse the second LL for each node of the first LL and if the same element is found create a new node.

2) Traverse both LL simultaneously and keep adding new nodes to a new LL if both are equal.

```
1.   Node* findIntersection(Node* head1, Node* head2)
2. {
3.  if(head1== NULL )
4.       return NULL;
```

```
5.    if(head2 == NULL)
6.        return NULL;
7.    int flag = 1;
8.    Node *temp;
9.    Node *new_head=NULL;
10.   while(head1 && head2)
11.   {
12.       if(head1->data == head2->data)
13.       {
14.           if(flag == 1)
15.           {
16.               new_head= new Node(head1->data);
17.               temp=new_head;
18.               temp->next = NULL;
19.               flag=0;
20.           }
21.           else
22.           {
23.               Node *newnode = new Node(head1->data);
24.               temp->next = newnode;
25.               newnode->next = NULL;
26.               temp=temp->next;
27.           }
28.           head1=head1->next;
29.           head2=head2->next;
30.       }
31.       else if(head1->data <head2->data)
32.       {
33.           head1=head1->next;
34.       }
```

```
35.      else if(head1->data > head2->data)
36.        {
37.        head2 = head2->next;
38.                }
39.  }
40.
41.  return new_head;
42.}
```

3)Implement (2) by recursion.

4) using a set:

```
1.    Node* findIntersection(Node* head1, Node* head2)
2.  {
3.  unordered_set<int> s;
4.    while(head2 != NULL)
5.    {
6.       s.insert(head2->data);
7.       head2=head2->next;
8.    }
9.    Node *ptr,*head=NULL;
10.   int flag=1;
11.   while(head1 != NULL)
12.   {
13.      if(s.find(head1->data)!=s.end())
14.      {
15.         if(flag)
16.         {
17.            flag=0;
18.            ptr=head1;
```

```
            head=head1;
        }
        else
        {
            ptr->next = head1;
            ptr = ptr->next;
        }

    }
    head1 = head1->next;
 }
 ptr->next = NULL;
 return head;
 // code here
 // return the head of intersection list
}
```

Union of Two Linked Lists

Given two linked lists, your task is to complete the function makeUnion(), that returns the union of two linked lists. This union should include all the distinct elements only.

Example :

Input:

L1 = 9->6->4->2->3->8

L2 = 1->2->8->6->2

Output: 1 2 3 4 6 8 9

Note: The new list formed should be in non-decreasing order.

Brute force Approach:

Initialize the result list as NULL. Traverse list1 and add all of its elements to the result.Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

2)using a set:

```
1. struct Node* makeUnion(struct Node* head1, struct Node* head2)
2. {
3.    Node *temp1 = head1 , *temp=head;
4.    while(temp1->next != NULL)
5.    {
6.       temp1 = temp1->next;
```

```
    }
    temp1->next = head2;

    Node *temp2 = temp,*ptr;
    unordered_set<int>s;
    while(temp2 != NULL )
    {
        if(s.find(temp2->data) == s.end())
        {
            s.insert(temp2->data);
            ptr = temp2;
            temp2 = temp2->next;
        }
        else
        {
            ptr->next = temp2->next;
            temp2 = temp2->next;
        }
    }
    return temp;
    // code here
}
```

Add two numbers represented by linked lists

Given two numbers represented by two linked lists of size **N** and **M**. The task is to return a sum list. The sum list is a linked list representation of the addition of two input numbers.

Example :

Input:

N = 2
valueN[] = {4,5}
M = 3
valueM[] = {3,4,5}

Output:
3 9 0

Approaches:

1)Iteratively traverse the List and keep adding the numbers from the linked list , if any of the list is finished then take its value as 0 and keep moving with the other list.

2)Recursively keep the value of carry and keep adding the nodes.

```
1.  Node *reverse(Node *head)
2.  {
3.     Node *curr = head , *nextp , *prevp = NULL;
4.     while(curr)
5.     {
6.        nextp = curr->next;
7.        curr->next = prevp;
8.        prevp = curr;
9.        curr = nextp;
10.    }
11.    return prevp;
12. }
13.
14. Node *addL(Node *head1 , Node *head2 , int &carry)
15. {
16.    int sum;
17.    if(head1 == NULL && head2 == NULL && carry == 0)
18.       return NULL;
19.
20.    else if(head1 == NULL && head2 == NULL)
21.    {
22.       Node*newnode = new Node(carry);
23.       newnode->next = NULL;
24.
25.       //Node *ptr = addL(head1->next , head2 , carry);
26.       //newnode->next = ptr;
27.
28.       return newnode;
29.    }
30.
```

```
31.  else if(head1 == NULL)
32.  {
33.      sum = head2->data + carry;
34.      carry = sum/10;
35.      sum = sum%10;
36.
37.      Node*newnode = new Node(sum);
38.      newnode->next = NULL;
39.
40.      Node *ptr = addL(head1, head2->next , carry);
41.      newnode->next = ptr;
42.
43.      return newnode;
44.  }
45.
46.  else if(head2 == NULL)
47.  {
48.
49.      sum = head1->data + carry;
50.      carry = sum/10;
51.      sum = sum%10;
52.
53.      Node*newnode = new Node(sum);
54.      newnode->next = NULL;
55.
56.      Node *ptr = addL(head1->next , head2 , carry);
57.      newnode->next = ptr;
58.
59.      return newnode;
60.  }
```

```
61.      sum = head1->data + head2->data + carry;
62.      carry = sum/10;
63.      sum = sum%10;
64.
65.
66.      Node*newnode = new Node(sum);
67.      newnode->next = NULL;
68.
69.      Node *ptr = addL(head1->next , head2->next , carry);
70.      newnode->next = ptr;
71.
72.      return newnode;
73.}
74.struct Node* addTwoLists(struct Node* first, struct Node* second)
75.{
76.  Node *head1 = reverse(first);
77.  Node *head2 = reverse(second);
78.
79.  int carry = 0;
80.  return reverse(addL(head1 , head2 , carry));
81.  // code here
82.}
```

3)Make two different functions , one for adding lists of the same size , other for adding lists of different sizes.

Questions:

Check-if-linked-list-is-palindrome

linked-list-of-strings-forms-a-palindrome

add-1-to-a-number-represented-as-linked-list

Add-two-numbers-represented-by-linked-lists

subtraction-in-linked-list

Intersection-of-two-sorted-linked-lists

intesertion of two linked list

Union-of-two-linked-list

Deletion-and-reverse-in-linked-list

Compare-two-linked-lists

delete-middle-of-linked-list

Decimal-equivalent-of-binary-linked-list