GeeksMan
Linked List
Lesson 4

Segregate even and odd nodes in a Linked List

Given a Linked List of integers, write a function to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list. Also, keep the order of even and odd numbers the same.

Input:
The first line of input contains an integer T denoting the number of test cases.The first line of each test case is N,N is the number of elements in the Linked List.The second line of each test case contains N input elements in Linked List.

Output:
Print the all even numbers then odd numbers in the modified Linked List.

Constraints:
1 ≤ T ≤ 100
1 ≤ N ≤ 100
1 ≤ size of elements ≤ 1000

Example:

Input

1
7
17 15 8 9 2 4 6

Output
8 2 4 6 17 15 9

Approach:

The idea is to split the linked list into two: one containing all even nodes and another containing all odd nodes. And finally attach the odd node linked list after the even node linked list.

https://sapphireengine.com/@/i6oemq

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

struct Node {
 int data;
 struct Node *next;
 Node(int x) {
   data = x;
   next = NULL;
 }
};

void print(Node *root)
{
Node *temp = root;
while(temp!=NULL)
{
cout<<temp->data<<" ";
temp=temp->next;
}
}
void segregateEvenOdd(Node **head_ref)
```

```
{
  Node *end = *head_ref;
  Node *prev = NULL;
  Node *curr = *head_ref;

  /* Get pointer to the last node */
  while (end->next != NULL)
     end = end->next;

  Node *new_end = end;


  while (curr->data % 2 != 0 && curr != end)
  {
     new_end->next = curr;
     curr = curr->next;
     new_end->next->next = NULL;
     new_end = new_end->next;
  }


  if (curr->data%2 == 0)
  {

     *head_ref = curr;


     while (curr != end)
     {
        if ( (curr->data) % 2 == 0 )
```

```
54.         {
55.             prev = curr;
56.             curr = curr->next;
57.         }
58.         else
59.         {
60.
61.             prev->next = curr->next;
62.             curr->next = NULL;
63.             new_end->next = curr;
64.             new_end = curr;
65.             curr = prev->next;
66.         }
67.     }
68. }
69. else prev = curr;
70.
71. if (new_end != end && (end->data) % 2 != 0)
72. {
73.     prev->next = end->next;
74.     end->next = NULL;
75.     new_end->next = end;
76. }
77. return;
78.}
79.
80.int main()
81.{
82.  int T;
83.         cin>>T;
```

```
84.
85.         while(T--)
86.         {
87.                 int K;
88.                 cin>>K;
89.                 struct Node *head = NULL;
90.       struct Node *temp = head;
91.
92.                 for(int i=0;i<K;i++){
93.                 int data;
94.                 cin>>data;
95.                         if(head==NULL)
96.                         head=temp=new Node(data);
97.                         else
98.                         {
99.                                 temp->next = new Node(data);
100.                                temp=temp->next;
101.                        }
102.                }
103.
104.
105.        segregateEvenOdd(&head);
106.        print(head);
107.        cout <<endl;
108.    }
109.        //code
110.        return 0;
111.    }
```

Arrange Consonants and Vowels

Given a singly linked list of size N containing only English Alphabets. Your task is to complete the function arrangeC&V(), that arranges the consonants and vowel nodes of the list in such a way that all the vowel nodes come before the consonants while maintaining the order of their arrival.

Input:

The function takes a single argument as input, the reference pointer to the head of the linked list. There will be T test cases and for each test case the function will be called separately.

Output:

For each test case output a single line containing space separated elements of the list.

Constraints:

1 <= T <= 100

1 <= N <= 100

Example:

Input:

2

6

a e g h i m

3

q r t

Output:

a e i g h m

q r t

Explanation:

Testcase 1: Vowels like a, e and i are in the front, and consonants like g, h and m are at the end of the list.

```
1. struct Node* arrange(Node *head)
2. 
3. {
4.       Node *curr = head , *v_head = NULL , *c_head = NULL  , *v_temp = NULL  , *c_temp = NULL ;
5. 
6.    int flagv = 1 , flagc = 1;
7. 
8.    while(curr!= NULL)
9.    {
10.        if(curr->data == 'a' || curr->data == 'e' || curr->data == 'i' || curr->data == 'o' || curr->data == 'u')
11. 
12.     {
13.         if(flagv)
14.         {
```

```
15.             v_head = curr;
16.             v_temp = curr;
17.             flagv = 0;
18.         }
19.
20.         else
21.         {
22.             v_temp->next = curr;
23.             v_temp = curr;
24.         }
25.     }
26.     else
27.     {
28.         if(flagc)
29.         {
30.             c_head = curr;
31.             c_temp = curr;
32.             flagc = 0;
33.         }
34.
35.         else
36.         {
37.             c_temp->next = curr;
38.             c_temp = curr;
39.         }
40.     }
41.
42.     curr = curr->next;
43. }
44. if(v_temp && c_temp)
```

```
45.  {
46.      v_temp -> next = c_head;
47.      c_temp->next = NULL;
48.      return v_head;
49.
50.  }
51.  if(v_temp == NULL)
52.  {
53.  return c_head;
54.  }
55.  if(c_temp == NULL)
56.  {
57.return v_head;
58.  }
59.  //Code here
60.}
```

Move all zeros to the front of the linked list

Given a linked list, the task is to move all 0's to the front of the linked list. The order of all other elements except 0 should be the same after rearrangement.

Input:

The first line of input contains an integer T denoting the number of test cases. For each test case, the first line contains an integer N denoting the number of elements in the Linked List and the second line contains N-space separated integer inputs i.e. elements in the Linked List.

Output:

For each test case, the output is the modified linked list having all 0's in front.

Constraints:

1 <= T <= 100

1 <= N <= 50

Note: List is from back to front.

Example:

Input:

2

10

0 1 0 1 2 0 5 0 4 0

7

0 0 0 3 2 1 1

Output:

0 0 0 0 0 4 5 2 1 1

0 0 0 1 1 2 3

Explanation:

1. Original list was 0->4->0->5->0->2->1->0->1->0->NULL.

   After processing the list becomes 0->0->0->0->0->4->5->2->1->1->NULL.

2. Original list was 1->1->2->3->0->0->0->NULL.

   After processing the list becomes 0->0->0->1->1->2->3->NULL.

```
1.  void moveZeroes(struct Node **head)
2.  {
3.     Node *temp = (*head)->next;
4.     Node *ptr = *(head);
5.     Node *curr;
6.     while(temp != NULL)
7.     {
8.        if(temp->data == 0)
9.        {
10.          curr = temp;
11.          temp = temp->next;
12.          curr->next = *(head);
13.          *(head) = curr;
14.          ptr->next = temp;
15.       }
16.       else
17.       {
18.          ptr = temp;
19.          temp = temp->next;
20.       }
21.    }
22.}
```

Given a linked list of 0s, 1s and 2s, sort it.

Given a linked list of N nodes where nodes can contain values 0s, 1s, and 2s only. The task is to segregate 0s, 1s, and 2s linked list such that all zeros segregate to the head side, 2s at the end of the linked list, and 1s in the mid of 0s and 2s.

Example 1:

Input:
N = 8
value[] = {1,2,2,1,2,0,2,2}
Output: 0 1 1 2 2 2 2 2
Explanation: All the 0s are segregated to the left end of the linked list, 2s to the right end of the list, and 1s in between.

Example 2:

Input:
N = 4
value[] = {2,2,0,1}
Output: 0 1 2 2
Explanation: After arranging all the 0s,1s and 2s in the given format, the output will be 0 1 2 2.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(1).

Constraints:

1 <= N <= 10^3

By taking count of each number :

```
1.  Node* segregate(Node *head)
2.  {
3.     int count[3] = {0,0,0};
4.        Node *temp = head;
5.        while(temp != NULL)
6.        {
7.           count[temp->data]++;
8.           temp = temp->next;
9.        }
10.       temp = head;
11.       for(int i = 0 ;i<3;i++)
12.       {
13.          while(count[i]>0)
14.          {
15.             temp->data = i;
16.             temp = temp->next;
17.             count[i]--;
18.          }
19.       }
20.       return head;
21.   // Add code here
22.
23.}
```

Another method is by changing the links Take dummy heads of each number and then in a single traversal,make 3 linked list connections and then connect the three.

Split Singly Linked List Alternatingly

Given a singly linked list of size N. Your task is to complete the function alternatingSplitList() that splits the given linked list into two smaller lists. The sublists should be made from alternating elements from the original list.

Note: the sublist should be in the order with respect to the original list.

Input Format:

First line of input contains the number of test cases T. First line of each input contains the length of the linked list and the next line contains the elements of the linked list.

Output Format:

For each test case, in new lines, print the two sublists formed after the split.

Constraints:

1 <= T <= 100

1 <= N <= 100

Example:

Input:

2

6

0 1 0 1 0 1

5

2 5 8 9 6

Output:

0 0 0

1 1 1

2 8 6

5 9

Explanation:

Testcase 1: After forming two sublists of the given list as required, we have two lists as: 0->0->0 and 1->1->1.

```
1.  void alternatingSplitList(struct Node* head)
2.  {
3.     if (head == NULL)
4.        return;
5.
6.     Node *odd = head;
7.     Node *even = head->next;
8.
9.     Node *evenFirst = even;
10.    Node *oddFirst = odd;
11.    while (true)
12.    {
13.       if ( !even || !(even->next))
14.       {
15.          odd->next = NULL;
16.          b = evenFirst;
17.          a = oddFirst;
18.          break;
19.       }
20.
21.       // Connecting odd nodes
```

```
22.      odd->next = even->next;
23.      odd = even->next;
24.
25.      if (odd->next == NULL)
26.      {
27.          even->next = NULL;
28.          odd->next = NULL;
29.          b = evenFirst;
30.          a = oddFirst;
31.          break;
32.      }
33.
34.      // Connecting even nodes
35.      even->next = odd->next;
36.      even = odd->next;
37.  //Your code here
38.}
39.}
```

Given a linked list, reverse alternate nodes and append at the end

Given a linked list, you have to perform the following task:

1. Extract the alternative nodes from starting from the second node.
2. Reverse the extracted list.
3. Append the extracted list at the end of the original list.

Example 1:

Input:
LinkedList = 10->4->9->1->3->5->9->4
Output: 10 9 3 9 4 5 1 4
Explanation: Reversing the alternative nodes from the given list, and then appending them to the end of the listresults in a list with the elements as
10 9 3 9 4 5 1 4.

Example 2:

Input:
LinkedList = 1->2->3->4->5
Output: 1 3 5 4 2
Explanation:

Note: Try to solve the problem without using any extra memory.

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(1)

Constraints:

1 <= N <= 105

0 <= Node_value <= 1000

```
1.  void rearrange(struct Node *odd)
2.  {
3.     //add code here
4.     Node *main=odd,*ev=odd->next,*second=odd->next;
5.     while(ev!=NULL &&  ev->next!=NULL)
6.     {
7.        main->next=main->next->next;
8.        main=main->next;
9.        ev->next=main->next;
10.       ev=ev->next;
11.    }
12.    main->next=NULL;
13.
14.
15.    Node *p=NULL,*c=second,*n;
16.    while(c!=NULL)
17.    {
18.       n=c->next;
19.       c->next=p;
20.       p=c;
21.       c=n;
22.    }
23.    main->next=p;
24.}
```

Modify Linked List-1

Given a singly linked list containing N nodes. Modify the value of first half nodes such that 1st node's new value is equal to the last node's value minus first node's current value, 2nd node's new value is equal to the second last node's value minus 2nd node's current value, likewise for first half nodes. If n is odd then the value of the middle node remains unchanged.

Note: Input in the linked list is like a new node will be entered at the head position (1st position).

Input:

First line consists of T test cases. First line of every test case consists of N, denoting the number of nodes. Second line of every test case consists of a Node of linked list.

Output:

Single line output, return the head of modified linked list.

Constraints:

1 <= T <= 200

1 <= N <= 100

Example:

Input:

2

5

10 4 5 3 6

6

2 9 8 12 7 10

Output:

-4 -1 5 4 10

8 -2 4 8 9 2

Explanation:

Testcase 1: After modifying the linked list as required, we have a new linked list containing the elements as -4, -1, 5, 4, 10.

```
1.  Node *rev(Node *head)
2.  {
3.    if(head==NULL || head->next==NULL)
4.    return head;
5.   Node*t= rev(head->next);
6.    head->next->next=head;
7.    head->next=NULL;
8.    return t;
9.  }
10.
11. struct Node* modifyTheList(struct Node *head)
12. {
13.      if(head == NULL || head->next == NULL)
14.        return head;
15.    Node *slow = head , *fast = head->next,*h=head;
16.    while(fast && fast->next)
17.    {
18.        slow = slow->next;
19.        fast = fast->next->next;
20.    }
21.
```

```
22.  fast=slow->next;
23.  slow->next=NULL;
24.  Node* r=rev(fast);
25.  Node* re=r;
26.  while(h)
27.  {
28.      h->data = h->data - r->data;
29.    r=r->next;
30.      h = h->next;
31.  }
32.  r=rev(re);
33.  slow->next=r;
34.
35.  return head;
```

split-a-circular-linked-list-into-two-halves=

Decimal-equivalent-of-binary-linked-list