Add 1 to a number represented as linked list

Number is represented in linked list such that each digit corresponds to a node in linked list. Add 1 to it. For example 1999 is represented as (1-> 9-> 9 -> 9) and adding 1 to it should change it to (2->0->0->0)

We strongly recommend that you click here and practice it, before moving on to the solution.

Below are the steps :

1. Reverse given linked list. For example, 1-> 9-> 9 -> 9 is converted to 9-> 9 -> 9 ->1.
2. Start traversing linked list from leftmost node and add 1 to it. If there is a carry, move to the next node. Keep moving to the next node while there is a carry.
3. Reverse modified linked list and return head.

Below is C++ implementation of above steps.

```
// C++ program to add 1 to a linked list
#include<bits/stdc++.h>

/* Linked list node */
struct Node
{
    int data;
    Node* next;
};

/* Function to create a new node with given data */
Node *newNode(int data)
{
    Node *new_node = new Node;
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

/* Function to reverse the linked list */
Node *reverse(Node *head)
{
    Node * prev  = NULL;
    Node * current = head;
    Node * next;
    while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

/* Adds one to a linked lists and return the head
   node of resultant list */
Node *addOneUtil(Node *head)
{
    // res is head node of the resultant list
    Node* res = head;
    Node *temp, *prev = NULL;

    int carry = 1, sum;
```

```
    while (head != NULL) //while both lists exist
    {
        // Calculate value of next digit in resultant list.
        // The next digit is sum of following things
        // (i) Carry
        // (ii) Next digit of head list (if there is a
        //     next digit)
        sum = carry + (head? head->data: 0);

        // update carry for next calulation
        carry = (sum >= 10)? 1 : 0;

        // update sum if it is greater than 10
        sum = sum % 10;

        // Create a new node with sum as data
        head->data = sum;

        // Move head and second pointers to next nodes
        temp = head;
        head = head->next;
    }

    // if some carry is still there, add a new node to
    // result list.
    if (carry > 0)
        temp->next = newNode(carry);

    // return head of the resultant list
    return res;
}

// This function mainly uses addOneUtil().
Node* addOne(Node *head)
{
    // Reverse linked list
    head = reverse(head);

    // Add one from left to right of reversed
    // list
    head = addOneUtil(head);

    // Reverse the modified list
    return reverse(head);
}

// A utility function to print a linked list
void printList(Node *node)
{
    while (node != NULL)
    {
        printf("%d", node->data);
        node = node->next;
    }
    printf("\n");
}

/* Driver program to test above function */
int main(void)
{
    Node *head = newNode(1);
    head->next = newNode(9);
    head->next->next = newNode(9);
    head->next->next->next = newNode(9);

    printf("List is ");
    printList(head);

    head = addOne(head);

    printf("\nResultant list is ");
    printList(head);
```

```
    return 0;
}
```

Output:

```
List is 1999

Resultant list is 2000
```

Recursive Implementation:
We can recursively reach the last node and forward carry to previous nodes. Recursive solution doesn't require reversing of linked list. We can also use a stack in place of recursion to temporarily hold nodes.

Below is C++ implementation of recursive solution.

```
// Recursive C++ program to add 1 to a linked list
#include<bits/stdc++.h>

/* Linked list node */
struct Node
{
    int data;
    Node* next;
};

/* Function to create a new node with given data */
Node *newNode(int data)
{
    Node *new_node = new Node;
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

// Recursively add 1 from end to beginning and returns
// carry after all nodes are processed.
int addWithCarry(Node *head)
{
    // If linked list is empty, then
    // return carry
    if (head == NULL)
        return 1;

    // Add carry returned be next node call
    int res = head->data + addWithCarry(head->next);

    // Update data and return new carry
    head->data = (res) % 10;
    return (res) / 10;
}

// This function mainly uses addWithCarry().
Node* addOne(Node *head)
{
    // Add 1 to linked list from end to beginning
    int carry = addWithCarry(head);

    // If there is carry after processing all nodes,
    // then we need to add a new node to linked list
    if (carry)
    {
        Node *newNode = new Node;
        newNode->data = carry;
        newNode->next = head;
        return newNode; // New node becomes head now
    }

    return head;
}
```

```
// A utility function to print a linked list
void printList(Node *node)
{
   while (node != NULL)
   {
      printf("%d", node->data);
      node = node->next;
   }
   printf("\n");
}

/* Driver program to test above function */
int main(void)
{
   Node *head = newNode(1);
   head->next = newNode(9);
   head->next->next = newNode(9);
   head->next->next->next = newNode(9);

   printf("List is ");
   printList(head);

   head = addOne(head);

   printf("\nResultant list is ");
   printList(head);

   return 0;
}
```

Output:

```
List is 1999

Resultant list is 2000
```