



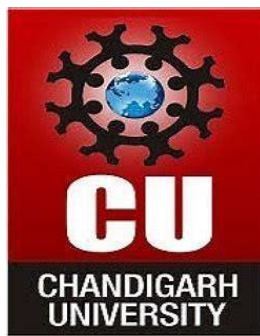
**CHANDIGARH  
UNIVERSITY**  
Discover. Learn. Empower.

**NAAC  
GRADE A+**  
Accredited University

## **UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**(BE-CSE)**



### **WINTER DOMAIN CAMP**

**Date : 23/12/2024**

**Submitted to:**

Faculty name: Er.Rajni Devi

**Submitted by:**

Name: Akshi Datta

UID: 22BCS15369

Section: IOT-620

## DAY 3(23/12/24)

### PROBLEM 1

**Objective:** The Fibonacci numbers, commonly denoted  $F(n)$  form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,  $F(0) = 0$ ,  $F(1) = 1$ ,  $F(n) = F(n - 1) + F(n - 2)$ , for  $n > 1$ . Given  $n$ , calculate  $F(n)$ .

#### Code:

```
#include <iostream>
using namespace std;
int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    int n;
    cout << "Enter a number (0 <= n <= 30): ";
    cin >> n;
    if (n < 0 || n > 30) {
        cout << "Please enter a number within the range 0 to 30." << endl;
        return 1;
    }
    cout << "F(" << n << ") = " << fibonacci(n) << endl;
    return 0;
}
```

#### Output:

```
Enter a number (0 <= n <= 30): 20
F(20) = 6765

=== Code Execution Successful ===
```

### PROBLEM 2

**Objective:** Write a program that returns the value of  $N!$  ( $N$  factorial) using recursion. Note that  $N! = 1 * 2 * \dots * N$ . Also,  $0! = 1$  and  $1! = 1$ .

#### Code:

```
#include <iostream>
```

```

using namespace std;
long long factorial(int n) {
    if (n == 0 || n == 1) return 1; // Base case: 0! = 1 and 1! = 1
    return n * factorial(n - 1);
}
int main() {
    int n;
    cout << "Enter a number (0 <= N <= 15): ";
    cin >> n;
    if (n < 0 || n > 15) {
        cout << "Please enter a number within the range 0 to 15." << endl;
        return 1;
    }
    cout << "The factorial of " << n << " is " << factorial(n) << endl;
    return 0;
}

```

### **Output:**

```

Enter a number (0 <= N <= 15): 10
The factorial of 10 is 3628800

```

```

=== Code Execution Successful ===

```

## **PROBLEM 3**

**Objective:** Given a number n, find sum of first n natural numbers. To calculate the sum, we will use a recursive function recur\_sum().

### **Code:**

```

#include <iostream>
using namespace std;
int recur_sum(int n) {
    if (n == 0) return 0;
    return n + recur_sum(n - 1);
}
int main() {
    int n;
    cout << "Enter a number (n >= 0): ";
    cin >> n;
    if (n < 0) {
        cout << "Please enter a non-negative number." << endl;
        return 1;
    }
    cout << "The sum of the first " << n << " natural numbers is " << recur_sum(n) << endl;
    return 0;
}

```

```
}
```

### **Output:**

```
Enter a number (n >= 0): 10
The sum of the first 10 natural numbers is 55

=== Code Execution Successful ===
```

## **PROBLEM 4**

**Objective:** Given an array of integers, find sum of array elements using recursion.

### **Code:**

```
#include <iostream>
using namespace std;
int array_sum(int A[], int n) {
    if (n == 0) return 0;
    return A[n - 1] + array_sum(A, n - 1);
}
int main() {
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int A[n];
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> A[i];
    }
    cout << "The sum of the array elements is " << array_sum(A, n) << endl;
    return 0;
}
```

### **Output:**

```
Enter the number of elements in the array: 7
Enter the elements of the array: 20 30 10 40 80 100 95
The sum of the array elements is 375

=== Code Execution Successful ===
```

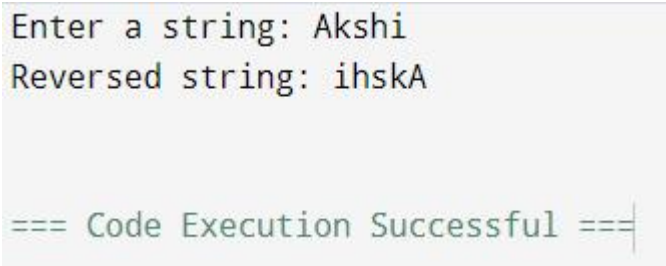
## **PROBLEM 5**

**Objective:** Write a recursive function to print the reverse of a given string.

### **Code:**

```
#include <iostream>
#include <string>
using namespace std;
string reverseString(string str) {
    int n = str.length();
    for (int i = 0; i < n / 2; i++) {
        swap(str[i], str[n - i - 1]);
    }
    return str;
}
int main() {
    string input;
    cout << "Enter a string: ";
    getline(cin, input);
    string reversed = reverseString(input);
    cout << "Reversed string: " << reversed << endl;
    return 0;
}
```

### **Output:**



```
Enter a string: Akshi
Reversed string: ihskA

=== Code Execution Successful ===
```

## **PROBLEM 6**

**Objective:** You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

### **Code:**

```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
```

```

ListNode* next;
ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
if (!list1) return list2;
if (!list2) return list1;
if (list1->val < list2->val) {
list1->next = mergeTwoLists(list1->next, list2);
return list1;
} else {
list2->next = mergeTwoLists(list1, list2->next);
return list2;
}
}
void printList(ListNode* head) {
while (head) {
cout << head->val << " ";
head = head->next;
}
cout << endl;
}
ListNode* createList(int arr[], int n) {
if (n == 0) return nullptr;
ListNode* head = new ListNode(arr[0]);
ListNode* current = head;
for (int i = 1; i < n; i++) {
current->next = new ListNode(arr[i]);
current = current->next;
}
return head;
}
int main() {
int arr1[] = {1, 2, 4};
int arr2[] = {1, 3, 4};
ListNode* list1 = createList(arr1, 3);
ListNode* list2 = createList(arr2, 3);
ListNode* mergedList = mergeTwoLists(list1, list2);
cout << "Merged List: ";
printList(mergedList);
return 0;
}

```

### **Output:**

```
Merged List: 1 1 2 3 4 4
```

```
=== Code Execution Successful ===
```

## **PROBLEM 7**

**Objective:** Given the head of a linked list and an integer val, remove all the nodes of the linked list that has `Node.val == val`, and return the new head.

### **Code:**

```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* removeElements(ListNode* head, int val) {
    if (!head) return nullptr;
    head->next = removeElements(head->next, val);
    return head->val == val ? head->next : head;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}
ListNode* createList(int arr[], int n) {
    if (n == 0) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < n; i++) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}
int main() {
    int arr[] = {1, 2, 6, 3, 4, 5, 6};
    int val = 6;
    ListNode* head = createList(arr, 7);
    cout << "Original List: ";
    printList(head);
    head = removeElements(head, val);
    cout << "List after removing " << val << ": ";
    printList(head);
    return 0;
}
```

## **Output:**

```
Output

Original List: 1 2 6 3 4 5 6
List after removing 6: 1 2 3 4 5

=== Code Execution Successful ===
```

## **PROBLEM 8**

**Objective:** Given the head of a singly linked list, reverse the list, and return the reversed list.

## **Code:**

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
Node* reverseList(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;
    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void printList(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
void appendNode(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
```



```

newNode->next = nullptr;
if (head == nullptr) {
    head = newNode;
} else {
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}
}
int main() {
    Node* head = nullptr;
    int n, value;
    cout << "Enter the number of elements in the list: ";
    cin >> n;
    cout << "Enter the elements of the list: ";
    for (int i = 0; i < n; i++) {
        cin >> value;
        appendNode(head, value);
    }
    cout << "Original List: ";
    printList(head);
    head = reverseList(head);
    cout << "Reversed List: ";
    printList(head);
    return 0;
}

```

### **Output:**

#### Output

```

Enter the number of elements in the list: 6
Enter the elements of the list: 90 60 52 87 30 111
Original List: 90 60 52 87 30 111
Reversed List: 111 30 87 52 60 90

```

## **PROBLEM 9**

**Objective:** Given an integer n, return true if it is a power of three. Otherwise, return false. An integer n is a power of three, if there exists an integer x such that  $n == 3^x$ .

### **Code:**

```

#include <iostream>
#include <cmath>

```

```

#include <climits>
using namespace std;
bool isPowerOfThree(int n) {
if (n <= 0) return false;
int maxPowerOfThree = pow(3, int(log(INT_MAX) / log(3)));
return maxPowerOfThree % n == 0;
}
int main() {
int n;
cout << "Enter an integer: ";
cin >> n;
if (isPowerOfThree(n)) {
cout << n << " is a power of three." << endl;
} else {
cout << n << " is not a power of three." << endl;
}
return 0;
}

```

### **Output:**

Output
<pre> Enter an integer: 3 3 is a power of three. </pre>

## **PROBLEM 10**

**Objective:** Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

### **Code:**

```

#include <iostream>
using namespace std;
struct ListNode {
int val;
ListNode* next;
ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* reverseList(ListNode* head) {
ListNode* prev = nullptr;
ListNode* curr = head;
while (curr) {
ListNode* nextNode = curr->next;
curr->next = prev;
prev = curr;
}
return prev;
}

```

```

curr = nextNode;
}
return prev;
}
bool isPalindrome(ListNode* head) {
if (!head || !head->next) return true;
ListNode *slow = head, *fast = head;
while (fast && fast->next) {
slow = slow->next;
fast = fast->next->next;
}
ListNode* secondHalf = reverseList(slow);
ListNode* firstHalf = head;
while (secondHalf) {
if (firstHalf->val != secondHalf->val) {
return false;
}
firstHalf = firstHalf->next;
secondHalf = secondHalf->next;
}
return true;
}
void printList(ListNode* head) {
while (head) {
cout << head->val << " ";
head = head->next;
}
cout << endl; }
ListNode* createList(int arr[], int n) {
if (n == 0) return nullptr;
ListNode* head = new ListNode(arr[0]);
ListNode* current = head;
for (int i = 1; i < n; i++) {
current->next = new ListNode(arr[i]);
current = current->next; }
return head; }
int main() {
int n;
cout << "Enter the number of elements in the linked list: ";
cin >> n;
int arr[n];
cout << "Enter the elements of the linked list: ";
for (int i = 0; i < n; i++) {
cin >> arr[i];
}
ListNode* head = createList(arr, n);
if (isPalindrome(head)) {
cout << "The linked list is a palindrome." << endl;
} else {
cout << "The linked list is not a palindrome." << endl;
}
}

```

```
return 0;
}
```

### **Output:**

#### Output

```
Enter the number of elements in the linked list: 3
Enter the elements of the linked list: 1 2 1
The linked list is a palindrome.
```

## **PROBLEM 12**

**Objective:** Alice and Bob are playing a game. Initially, Alice has a string word = "a". You are given a positive integer k. Now Bob will ask Alice to perform the following operation forever: Generate a new string by changing each character in word to its next character in the English alphabet, and append it to the original word.

For example, performing the operation on "c" generates "cd" and performing the operation on "zb" generates "zbac". Return the value of the kth character in word, after enough operations have been done for word to have at least k characters. Note that the character 'z' can be changed to 'a' in the operation.

### **Code:**

```
#include <iostream>
#include <string>
using namespace std;
char findKthCharacter(int k) {
    string word = "a";
    int len = 1;
    while (len < k) {
        string nextPart = "";
        for (char c : word) {
            nextPart += (c == 'z') ? 'a' : c + 1;
        }
        word += nextPart;
        len = word.length();
    }
    return word[k - 1];
}
int main() {
    int k;
    cout << "Enter the value of k: ";
    cin >> k;
    cout << "The " << k << "th character is: " << findKthCharacter(k) << endl;
    return 0;
}
```

## **Output:**

### Output

```
Enter the value of k: 6
The 6th character is: c
```

## **PROBLEM 13**

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

## **Code:**

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> sumArrays(const vector<int>& a1, const vector<int>& a2) {
    vector<int> result;
    int size = min(a1.size(), a2.size()); // Ensure the arrays align by size
    for (int i = 0; i < size; i++) {
        result.push_back(a1[i] + a2[i]);
    }
    return result;
}
int main() {
    int n1, n2;
    cout << "Enter the number of elements for the first array (a1): ";
    cin >> n1;
    vector<int> a1(n1);
    cout << "Enter the elements of the first array (a1): ";
    for (int i = 0; i < n1; i++) {
        cin >> a1[i];
    }
    cout << "Enter the number of elements for the second array (a2): ";
    cin >> n2;
    vector<int> a2(n2);
    cout << "Enter the elements of the second array (a2): ";
    for (int i = 0; i < n2; i++) {
        cin >> a2[i];
    }
    vector<int> result = sumArrays(a1, a2);
```

```

cout << "Output array: [";
for (size_t i = 0; i < result.size(); i++) {
    cout << result[i];
    if (i < result.size() - 1) {
        cout << ", ";
    }
}
cout << "]" << endl;
return 0;
}

```

### **Output:**

```

Output
Output array: [7, 8, 8]

```

## **PROBLEM 14**

**Objective:** You have a list arr of all integers in the range [1, n] sorted in a strictly increasing order. Apply the following algorithm on arr: Starting from left to right, remove the first number and every other number afterward until you reach the end of the list. Repeat the previous step again, but this time from right to left, remove the rightmost number and every other number from the remaining numbers. Keep repeating the steps again, alternating left to right and right to left, until a single number remains. Given the integer n, return the last number that remains in arr.

### **Code:**

```

#include <iostream>
#include <cmath>
using namespace std;
int lastRemaining(int n) {
    int largestPowerOf2 = 1;
    while (largestPowerOf2 <= n) {
        largestPowerOf2 *= 2;
    }
    largestPowerOf2 /= 2;
    return 2 * (n - largestPowerOf2) + 1;
}
int main() {
    int n;
    cout << "Enter the value of n: ";
    cin >> n;
    cout << "The last remaining number is: " << lastRemaining(n) << endl;
    return 0;
}

```

## Output:

### Output

```
Enter the value of n: 5
The last remaining number is: 3
```

## PROBLEM 15

**Objective:** You are given an integer array nums. Two players are playing a game with this array: player 1 and player 2. Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of 0. At each turn, the player takes one of the numbers from either end of the array (i.e., nums[0] or nums[nums.length - 1]) which reduces the size of the array by 1. The player adds the chosen number to their score. The game ends when there are no more elements in the array. Return true if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return true. You may assume that both players are playing optimally.

## Code:

```
#include <iostream>
#include <vector>
using namespace std;
bool PredictTheWinner(vector<int>& nums) {
    int n = nums.size();
    vector<vector<int>> dp(n, vector<int>(n, 0));
    for (int i = 0; i < n; i++) {
        dp[i][i] = nums[i];
    }
    for (int length = 2; length <= n; length++) {
        for (int i = 0; i <= n - length; i++) {
            int j = i + length - 1;
            dp[i][j] = max(nums[i] - dp[i+1][j], nums[j] - dp[i][j-1]);
        }
    }
    return dp[0][n-1] >= 0;
}
int main() {
    vector<int> nums1 = {1, 5, 2};
    vector<int> nums2 = {1, 5, 233, 7};
    cout << "Test Case 1: " << (PredictTheWinner(nums1) ? "True" : "False") << endl;
    cout << "Test Case 2: " << (PredictTheWinner(nums2) ? "True" : "False") << endl;
    return 0;
}
```

### **Output:**

Output

Test Case 1: False

Test Case 2: True

### **PROBLEM 16**

**Objective:** There are  $n$  friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to  $n$  in clockwise order. More formally, moving clockwise from the  $i$ th friend brings you to the  $(i+1)$ th friend for  $1 \leq i < n$ , and moving clockwise from the  $n$ th friend brings you to the 1st friend.

The rules of the game are as follows: Start at the 1st friend. Count the next  $k$  friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once. The last friend you counted leaves the circle and loses the game. If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat. Else, the last friend in the circle wins the game. Given the number of friends,  $n$ , and an integer  $k$ , return the winner of the game.

### **Code:**

```
#include <iostream>
using namespace std;
int findTheWinner(int n, int k) {
    int winner = 0;
    for (int i = 2; i <= n; i++) {
        winner = (winner + k) % i;
    }
    return winner + 1;
}
int main() {
    int n1 = 5, k1 = 2;
    int n2 = 6, k2 = 5;
    cout << "Winner for n = " << n1 << ", k = " << k1 << ": " << findTheWinner(n1, k1) << endl;
    cout << "Winner for n = " << n2 << ", k = " << k2 << ": " << findTheWinner(n2, k2) << endl;
    return 0;
}
```

### **Output:**



### Output

```
Winner for n = 5, k = 2: 3
Winner for n = 6, k = 5: 1
```

## **PROBLEM 17**

**Objective:** You are given a positive integer  $p$ . Consider an array `nums` (1-indexed) that consists of the integers in the inclusive range  $[1, 2p - 1]$  in their binary representations. You are allowed to do the following operation any number of times: Choose two elements  $x$  and  $y$  from `nums`. Choose a bit in  $x$  and swap it with its corresponding bit in  $y$ . Corresponding bit refers to the bit that is in the same position in the other integer. For example, if  $x = 1101$  and  $y = 0011$ , after swapping the 2nd bit from the right, we have  $x = 1111$  and  $y = 0001$ . Find the minimum non-zero product of `nums` after performing the above operation any number of times. Return this product modulo  $10^9 + 7$ .

### **Code:**

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
const int MOD = 1e9 + 7;
long long minimumProduct(int p) {
    int n = (1 << p) - 1;
    vector<int> nums;
    for (int i = 1; i <= n; ++i) {
        nums.push_back(i);
    }
    long long productModulo = 1;
    for (int i = 0; i < nums.size(); ++i) {
        productModulo = (productModulo * nums[i]) % MOD;
    }
    return productModulo;
}
int main() {
    int p;
    cout << "Enter value of p: ";
    cin >> p;
    cout << "Minimum product is: " << minimumProduct(p) << endl;
    return 0;
}
```

### **Output:**

## Output

```
Enter value of p: 10
Minimum product is: 942371722
```

## PROBLEM 18

**Objective:** Given an input string *s* and a pattern *p*, implement regular expression matching with support for '.' and '\*' where:

'.' Matches any single character.

'\*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

### Code:

```
#include <iostream>
#include <vector>
using namespace std;
bool isMatch(string s, string p) {
    int m = s.length(), n = p.length();
    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));
    dp[0][0] = true;
    for (int j = 1; j <= n; j++) {
        if (p[j - 1] == '*' && dp[0][j - 2]) {
            dp[0][j] = true;
        }
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (s[i - 1] == p[j - 1] || p[j - 1] == '.') {
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                dp[i][j] = dp[i][j - 2] || (dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'));
            }
        }
    }
    return dp[m][n];
}
int main() {
    string s, p;
    cout << "Enter string s: ";
    cin >> s;
    cout << "Enter pattern p: ";
    cin >> p;
    if (isMatch(s, p)) {
        cout << "Match successful!" << endl;
    }
}
```

```

} else {
cout << "No match." << endl;
}
return 0;
}

```

### **Output:**

```

Output

Enter string s: aa
Enter pattern p: a*
Match successful!

```

## **PROBLEM 19**

**Objective:** Given the head of a linked list, reverse the nodes of the list k at a time, and return the modified list. k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is. You may not alter the values in the list's nodes, only nodes themselves may be changed.

### **Code:**

```

#include <iostream>
using namespace std;

struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

ListNode* reverseKGroup(ListNode* head, int k) {
    int length = 0;
    ListNode* current = head;
    while (current != NULL) {
        length++;
        current = current->next;
    }

    ListNode* dummy = new ListNode(0);
    dummy->next = head;

```

```

ListNode* prevGroupEnd = dummy;

while (length >= k) {
    ListNode* groupStart = prevGroupEnd->next;
    ListNode* groupEnd = groupStart;
    for (int i = 1; i < k; i++) {
        groupEnd = groupEnd->next;
    }

    ListNode* nextGroupStart = groupEnd->next;
    ListNode* prev = nextGroupStart;
    ListNode* curr = groupStart;

    while (curr != nextGroupStart) {
        ListNode* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;
    }

    prevGroupEnd->next = groupEnd;
    groupStart->next = nextGroupStart;

    prevGroupEnd = groupStart;

    length -= k;
}

return dummy->next;
}

void printList(ListNode* head) {
    ListNode* current = head;
    while (current != NULL) {
        cout << current->val << " ";
        current = current->next;
    }
    cout << endl;
}

int main() {
    ListNode* head1 = new ListNode(1);
    head1->next = new ListNode(2);
    head1->next->next = new ListNode(3);
    head1->next->next->next = new ListNode(4);
    head1->next->next->next->next = new ListNode(5);
    int k1 = 2;
    ListNode* result1 = reverseKGroup(head1, k1);
    printList(result1);

    ListNode* head2 = new ListNode(1);

```

```
head2->next = new ListNode(2);
head2->next->next = new ListNode(3);
head2->next->next->next = new ListNode(4);
head2->next->next->next->next = new ListNode(5);
int k2 = 3;
ListNode* result2 = reverseKGroup(head2, k2);
printList(result2);
return 0;
}
```

### **Output:**

Output					
2	1	4	3	5	
3	2	1	4	5	