DOMAIN WINTER CAMP

Department of Computer Science and Engineering

Name:- Sarvagya Pathak UID:- 22BCS10032 Section:- 22KPIT-901-A

DAY-3

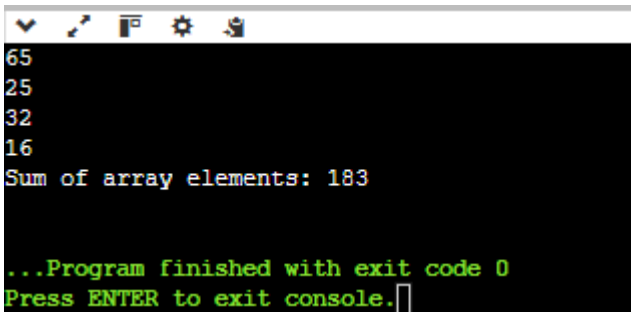
Q.1. Given an array of integers, find sum of array elements using recursion.

Program Code:-

```
#include <iostream>

using namespace std;

// Recursive function to calculate the sum of array elements

int sumArray(int arr[], int n) {

    // Base case: If the size of the array is 0, the sum is 0

    if (n == 0) {

        return 0;

    }

    // Recursive case: Add the last element to the sum of the remaining array

    return arr[n - 1] + sumArray(arr, n - 1);

}int main() {
```

```
    int n
    // Input the size of the array
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[n]
    // Input the elements of the array
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    // Calculate and print the sum of the array elements
    int result = sumArray(arr, n);
    cout << "Sum of array elements: " << result << endl;
    return 0;
```

Output:-



Q.2. Given the head of a singly linked list, reverse the list, and return the reversed list.

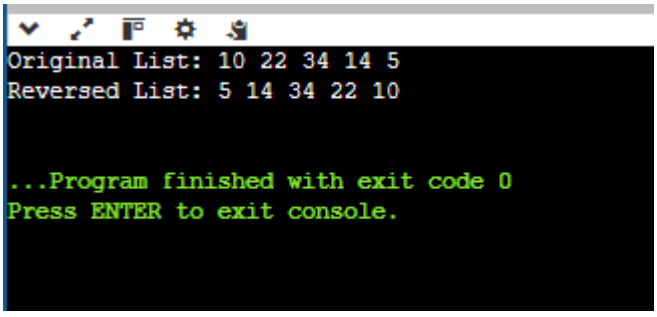
Program Code:-

```
#include <iostream>

using namespace std;

// Definition for singly-linked list node

struct ListNode {

    int val;

    ListNode* next;

    ListNode(int x) : val(x), next(nullptr) {}

};

// Function to reverse the singly linked list

ListNode* reverseList(ListNode* head) {

    ListNode* prev = nullptr; // Previous pointer

    ListNode* current = head; // Current pointer

    while (current != nullptr) {

        ListNode* nextNode = current->next; // Save the next node

        current->next = prev; // Reverse the link

        prev = current; // Move prev forward
```

```
        current = nextNode; // Move current forward
    }
    return prev; // New head of the reversed list
}
// Function to print the linked list
void printList(ListNode* head) {
    ListNode* temp = head;
    while (temp != nullptr) {
        cout << temp->val << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Main function
int main() {
    // Create a linked list: 1 -> 2 -> 3 -> 4 -> 5
    ListNode* head = new ListNode(1);
    head->next = new ListNode(2);
```

```
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
    head->next->next->next->next = new ListNode(5);
    cout << "Original List: ";
    printList(head);
    // Reverse the list
    ListNode* reversedHead = reverseList(head);
    cout << "Reversed List: ";
    printList(reversedHead);
    return 0;
}
```

Output:-



Q.3. 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.

Program Code:-

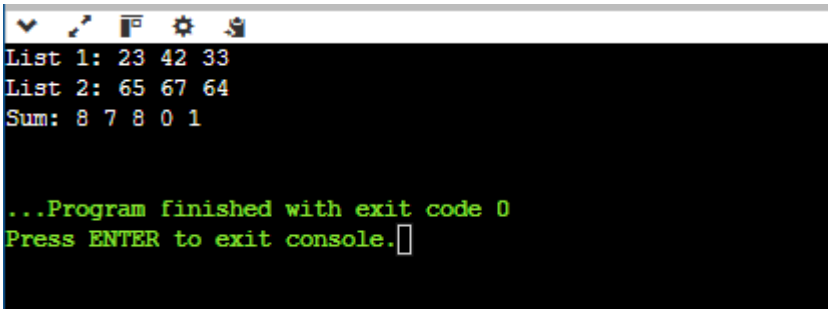
```
#include <iostream>
using namespace std;

// Definition for singly-linked list node
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to add two numbers represented as linked lists
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode* dummyHead = new ListNode(0); // Dummy node to simplify the logic
    ListNode* current = dummyHead;
    int carry = 0;

    // Traverse both lists
```

```
    while (l1 != nullptr || l2 != nullptr || carry != 0) {
        int sum = carry;
        if (l1 != nullptr) {
            sum += l1->val;
            l1 = l1->next;
        }
        if (l2 != nullptr) {
            sum += l2->val;
            l2 = l2->next;
        }

        carry = sum / 10; // Compute carry
        current->next = new ListNode(sum % 10); // Create a new node for the current digit
        current = current->next;
    }

    return dummyHead->next; // Return the next node of the dummy head
}
```

```
// Function to print the linked list
void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

// Main function
int main() {
    // Create the first linked list: 2 -> 4 -> 3 (342)
    ListNode* l1 = new ListNode(23);
    l1->next = new ListNode(42);
    l1->next->next = new ListNode(33);

    // Create the second linked list: 5 -> 6 -> 4 (465)
    ListNode* l2 = new ListNode(65);
    l2->next = new ListNode(67);
```

```
    l2->next->next = new ListNode(64);

    cout << "List 1: ";
    printList(l1);

    cout << "List 2: ";
    printList(l2);

    // Add the two numbers
    ListNode* result = addTwoNumbers(l1, l2);

    cout << "Sum: ";
    printList(result);

    return 0;
}
```

Output:-



Q.4. 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.

Program Code:-

```
#include <iostream>

using namespace std;


// Definition for singly-linked list node

struct ListNode {

    int val;

    ListNode* next;

    ListNode(int x) : val(x), next(nullptr) {}
```

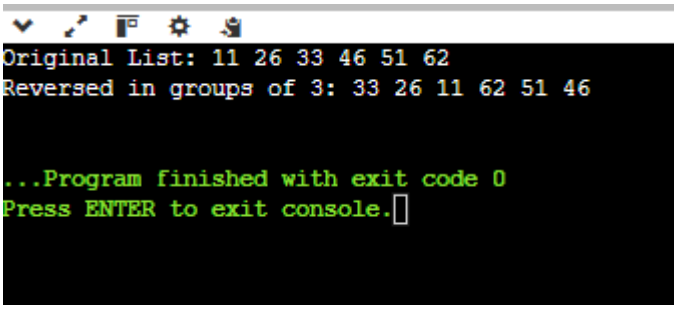
```
};

// Function to reverse a sublist of k nodes
ListNode* reverseKNodes(ListNode* head, int k) {
    ListNode* prev = nullptr;
    ListNode* current = head;
    ListNode* nextNode = nullptr;
    int count = 0;

    // Reverse k nodes
    while (current != nullptr && count < k) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
        count++;
    }

    // Return the new head of the reversed sublist
```

```
    return prev;
}

// Function to reverse nodes in k-sized groups
ListNode* reverseKGroup(ListNode* head, int k) {
    if (!head || k == 1) return head;

    // Check if there are at least k nodes left
    ListNode* temp = head;
    int count = 0;
    while (temp != nullptr && count < k) {
        temp = temp->next;
        count++;
    }

    // If there are at least k nodes, reverse them
    if (count == k) {
        ListNode* reversedHead = reverseKNodes(head, k);
        head->next = reverseKGroup(temp, k); // Recursively reverse the rest of the list
```

```
        return reversedHead;
    }

    // If fewer than k nodes are left, return the head as is
    return head;
}

// Function to print the linked list
void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

// Main function
int main() {
    // Create a linked list: 1 -> 2 -> 3 -> 4 -> 5 -> 6
```

```
    ListNode* head = new ListNode(11);
    head->next = new ListNode(26);
    head->next->next = new ListNode(33);
    head->next->next->next = new ListNode(46);
    head->next->next->next->next = new ListNode(51);
    head->next->next->next->next->next = new ListNode(62);

    cout << "Original List: ";
    printList(head);

    int k = 3; // Group size
    ListNode* result = reverseKGroup(head, k);

    cout << "Reversed in groups of " << k << ": ";
    printList(result);

    return 0;
}
```

Output:-



Q.5. You are given a positive integer primeFactors. You are asked to construct a positive integer n that satisfies the following conditions:

The number of prime factors of n (not necessarily distinct) is at most primeFactors.

The number of nice divisors of n is maximized. Note that a divisor of n is nice if it is divisible by every prime factor of n. For example, if n = 12, then its prime factors are [2,2,3], then 6 and 12 are nice divisors, while 3 and 4 are not.

Return the number of nice divisors of n. Since that number can be too large, return it modulo 109 + 7.

Note that a prime number is a natural number greater than 1 that is not a product of two smaller natural numbers. The prime factors of a number n is a list of prime numbers such that their product equals n.

Program Code:-

#include <iostream>

#include <cmath>

using namespace std;

const int MOD

=1e9+7; long long

powerMod(long long

```
base, long long exp,
long long mod) {
long long result = 1;
while (exp > 0) {
if (exp % 2 == 1) {
result = (result * base)
% mod;
        }
        base = (base * base) % mod;
exp /= 2;
    }     return
result;
} int maxNiceDivisors(int
primeFactors) {     if (primeFactors ==
1) return 1;     int q = primeFactors / 3;
int r = primeFactors % 3;

    if (r == 0) {  return powerMod(3,
q, MOD);        } else if (r == 1) {
```

```
return (powerMod(3,q - 1, MOD) *
4) %MOD;
    } else {
        return (powerMod(3, q, MOD) * 2) % MOD;
    }
} int main() {     int primeFactors;
   cout << "Enter the number of prime factors: ";
   cin >> primeFactors;
    cout << "Number of nice divisors: " << maxNiceDivisors(primeFactors) <<
endl;
return 0; }
```

Output:-