CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

DOMAIN WINTER CAMP

(Department of Computer Science and Engineering)

Name: Ankit Vashisth UID: 22BCS13378 Section: KPIT 901-B

DAY 3

(Easy)

Q1. The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two prece ding ones, starting from 0 and 1. That is,

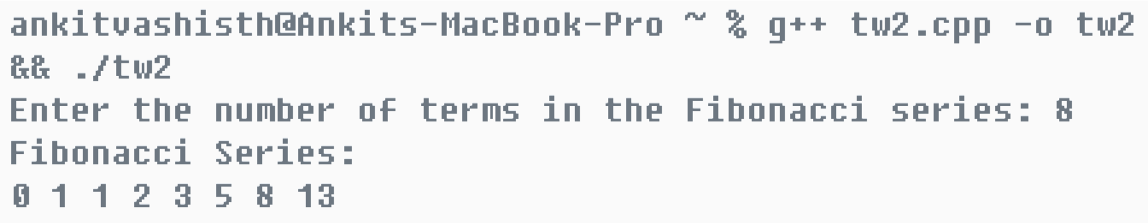
F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

Program code:

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

// Function to calculate Fibonacci numbers using recursion
int fibonacci(int n) {
   if (n <= 1)
      return n;
   return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
   int n;
   cout << "Enter the number of terms in the Fibonacci series: ";
   cin >> n;
     cout << "Fibonacci Series: \n";
   for (int i = 0; i < n; i++) {
      cout << fibonacci(i) << " ";
   }
   cout << endl;


   return 0;
}
```

Output:



```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ tw2.cpp -o tw2
&& ./tw2
Enter the number of terms in the Fibonacci series: 8
Fibonacci Series:
0 1 1 2 3 5 8 13
```

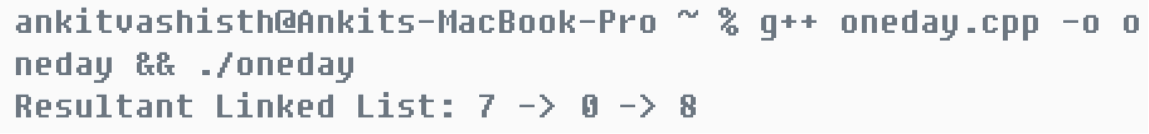
Q 2 (Medium)You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

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

Program code:

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

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

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

    while (l1 != nullptr || l2 != nullptr || carry != 0) {
        int sum = carry; // Start with the carry from the previous operation
        if (l1 != nullptr) {
            sum += l1->val;
            l1 = l1->next;
        }
        if (l2 != nullptr) {
            sum += l2->val;
            l2 = l2->next;
        }

        carry = sum / 10; // Update carry
        current->next = new ListNode(sum % 10); // Add the digit to the result
        current = current->next; // Move to the next node
    }

    return dummyHead->next; // Return the result list, skipping the dummy head
}

// Helper function to create a linked list from a vector of integers
ListNode* createList(const vector<int>& nums) {
    ListNode* dummyHead = new ListNode();
    ListNode* current = dummyHead;
    for (int num : nums) {
        current->next = new ListNode(num);
        current = current->next;
    }
    return dummyHead->next;
}

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

int main() {
    // Input lists
    vector<int> list1 = {2, 4, 3};
    vector<int> list2 = {5, 6, 4};

    // Create linked lists
    ListNode* l1 = createList(list1);
    ListNode* l2 = createList(list2);

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

    // Print the result
    cout << "Resultant Linked List: ";
    printList(result);

    return 0;
}
```

Output:



```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ oneday.cpp -o o
neday && ./oneday
Resultant Linked List: 7 -> 0 -> 8
```

Ques 3. 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.

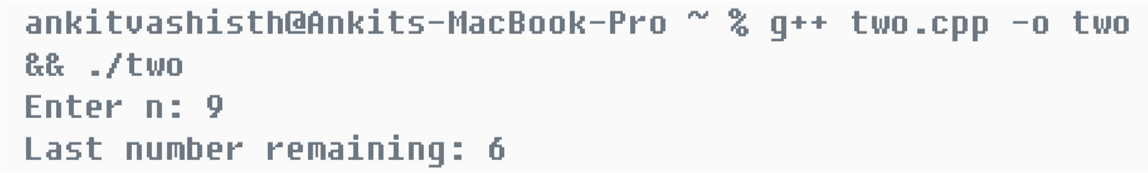
Given the integer n, return the last number that remains in arr.

Program Code:

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

int lastRemaining(int n) {
   int start = 1;      // Start of the list
   int step = 1;       // Step size for elimination
   int remaining = n;  // Remaining numbers
   bool leftToRight = true; // Direction of elimination

   while (remaining > 1) {
      // If eliminating from left or the number of remaining elements is odd
      if (leftToRight || remaining % 2 == 1) {
         start += step;
      }
      // Update step and remaining numbers
      step *= 2;
      remaining /= 2;
      leftToRight = !leftToRight; // Alternate direction
   }

   return start;
}
```

```
int main() {
   int n;
   cout << "Enter n: ";
   cin >> n;

   cout << "Last number remaining: " << lastRemaining(n) << endl;
   return 0;
}
```

Output:



```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ two.cpp -o two
&& ./two
Enter n: 9
Last number remaining: 6
```

Ques 4. Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

'.' Matches any single character.

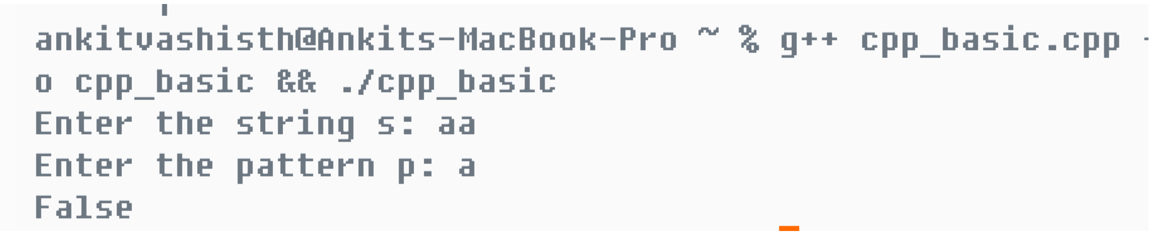
'*' Matches zero or more of the preceding element.

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

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

bool isMatch(string s, string p) {
   int m = s.size(), n = p.size();
   // Create a DP table
   vector<vector<bool>> dp(m + 1, vector<bool>(n + 1,
false));
   dp[0][0] = true; // Both strings are empty

   // Handle patterns with '*'
   for (int j = 1; j <= n; j++) {
      if (p[j - 1] == '*') {
         dp[0][j] = dp[0][j - 2]; // '*' matches zero
occurrences
```

```
        }
    }

    // Fill the DP table
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '.') {
                // Characters match, or '.' matches any character
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                // '*' matches zero occurrences or one/more of
the preceding character
                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 the string s: ";
    cin >> s;
    cout << "Enter the pattern p: ";
    cin >> p;

    if (isMatch(s, p)) {
        cout << "True" << endl;
    } else {
        cout << "False" << endl;
    }

    return 0;
}
```

Output:



```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ cpp_basic.cpp -
o cpp_basic && ./cpp_basic
Enter the string s: aa
Enter the pattern p: a
False
```

Ques 5 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.

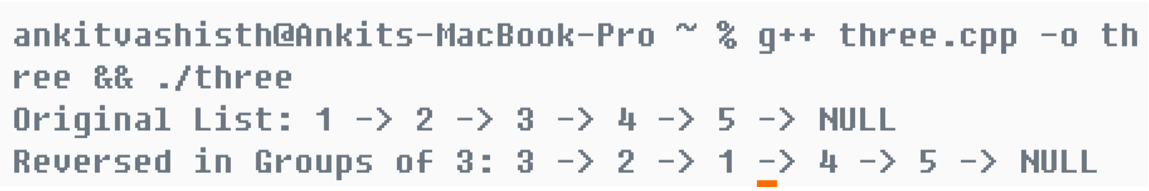
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Program Code:

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

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

// Function to reverse a portion of the linked list
ListNode* reverse(ListNode* head, ListNode* tail) {
   ListNode* prev = nullptr;
   ListNode* curr = head;
   while (curr != tail) {
      ListNode* nextTemp = curr->next;
      curr->next = prev;
      prev = curr;
      curr = nextTemp;
   }
   return prev; // New head of the reversed list
```

```
}

// Function to reverse k nodes at a time
ListNode* reverseKGroup(ListNode* head, int k) {
    ListNode* dummy = new ListNode(0);
    dummy->next = head;
    ListNode* prevGroupEnd = dummy;

    while (true) {
        // Check if there are k nodes left to reverse
        ListNode* groupStart = prevGroupEnd->next;
        ListNode* groupEnd = prevGroupEnd;
        for (int i = 0; i < k; i++) {
            groupEnd = groupEnd->next;
            if (!groupEnd) return dummy->next; // Not enough
nodes left
        }

        // Reverse the group
        ListNode* nextGroupStart = groupEnd->next;
        ListNode* newGroupHead = reverse(groupStart,
groupEnd->next);

        // Connect the reversed group to the previous and
next parts
        prevGroupEnd->next = newGroupHead;
        groupStart->next = nextGroupStart;

        // Move to the next group
        prevGroupEnd = groupStart;
    }
}

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

int main() {
    // Creating the 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);

    int k = 3;
    cout << "Original List: ";
    printList(head);

    ListNode* result = reverseKGroup(head, k);

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

    return 0;
}
```

Output:



```
ankitvashisth@Ankits-MacBook-Pro ~ % g++ three.cpp -o th
ree && ./three
Original List: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Reversed in Groups of 3: 3 -> 2 -> 1 -> 4 -> 5 -> NULL
```