Department of Computer Science and Engineering

Name:- Bhavya Gupta   UID:- 22BCS10246   Section:- 22KPIT-901-A

DAY-3

Q.1. **Fibonacci Series Using Recursion**

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).

Example 1:

Input: n = 2

Output: 1

Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1

Example 2:

Input: n = 3

Output: 2

Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.

Constraints:
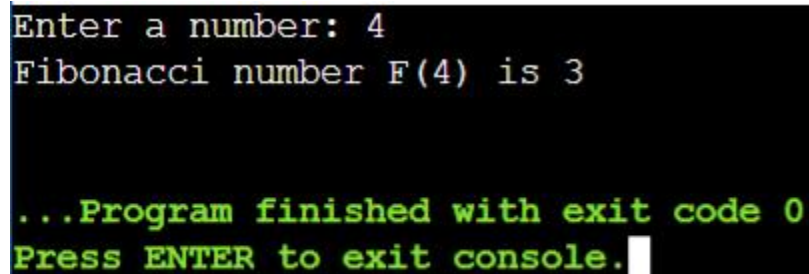
$0 <= n <= 30$

Program Code:-

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

int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    cout << "Fibonacci number F(" << n << ") is " << fibonacci(n) << endl;
    return 0;
}
```

Output:-

```
Enter a number: 4
Fibonacci number F(4) is 3


...Program finished with exit code 0
Press ENTER to exit console.
```

Reverse Linked List

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

Example 1:

Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2:

Input: head = [1,2]

Output: [2,1]

Constraints:

The number of nodes in the list is the range [0, 5000].

-5000 <= Node.val <= 5000

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

Program Code:-

```cpp
#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;
```

```cpp
    ListNode* curr = head;
    ListNode* next = nullptr;

    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    return prev;
}

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

ListNode* createNode(int value) {
    return new ListNode(value);
}

int main() {
    ListNode* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = createNode(5);

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

    ListNode* reversedHead = reverseList(head);

    cout << "Reversed Linked List: ";
    printList(reversedHead);

    return 0;
```

}

Output:-

```
Original Linked List: 1 2 3 4 5
Reversed Linked List: 5 4 3 2 1


...Program finished with exit code 0
Press ENTER to exit console.
```

Q3. **Add Two Numbers**

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.

Example 1:

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

Example 2:

 Input: l1 = [0], l2 = [0]

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Constraints:

 The number of nodes in each linked list is in the range [1, 100].

0 <= Node.val <= 9

It is guaranteed that the list represents a number that does not have leading zeros.

Program Code:-

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

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

ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode* dummyHead = new ListNode(0);
    ListNode* current = dummyHead;
    int carry = 0;

    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;
```

```cpp
            current->next = new ListNode(sum % 10);
            current = current->next;
        }

        return dummyHead->next;
    }

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

    ListNode* createNode(int value) {
        return new ListNode(value);
    }

    int main() {
        ListNode* l1 = createNode(2);
        l1->next = createNode(4);
        l1->next->next = createNode(3);

        ListNode* l2 = createNode(5);
        l2->next = createNode(6);
        l2->next->next = createNode(4);

        ListNode* result = addTwoNumbers(l1, l2);
        cout << "Result Linked List: ";
        printList(result);

        ListNode* l3 = createNode(0);
        ListNode* l4 = createNode(0);
        ListNode* result2 = addTwoNumbers(l3, l4);
        cout << "Result Linked List: ";
        printList(result2);

        ListNode* l5 = createNode(9);
        l5->next = createNode(9);
```

```cpp
    l5->next->next = createNode(9);
    l5->next->next->next = createNode(9);
    l5->next->next->next->next = createNode(9);
    l5->next->next->next->next->next = createNode(9);
    l5->next->next->next->next->next->next = createNode(9);

    ListNode* l6 = createNode(9);
    l6->next = createNode(9);
    l6->next->next = createNode(9);
    l6->next->next->next = createNode(9);

    ListNode* result3 = addTwoNumbers(l5, l6);
    cout << "Result Linked List: ";
    printList(result3);

    return 0;
}
```

Output:-

```
Result Linked List: 7 0 8
Result Linked List: 0
Result Linked List: 8 9 9 9 0 0 0 1


...Program finished with exit code 0
Press ENTER to exit console.
```

Q4. **Wildcard Matching**

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

'?' Matches any single character.

'*' Matches any sequence of characters (including the empty sequence).

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

Example 1:

Input: s = "aa", p = "a"

Output: false

Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p = "*"

Output: true

Explanation: '*' matches any sequence.

Example 3:

Input: s = "cb", p = "?a"

Output: false

Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

Constraints:

0 <= s.length, p.length <= 2000

s contains only lowercase English letters.

p contains only lowercase English letters, '?' or '*'.

Program Code:-

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

```cpp
bool isMatch(string s, string p) {
    int m = s.length();
    int 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] = dp[0][j - 1];
        }
    }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '?') {
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
            }
        }
    }

    return dp[m][n];
}

int main() {
    cout << boolalpha;
    cout << "Test Case 1: " << isMatch("aa", "a") << endl;
    cout << "Test Case 2: " << isMatch("aa", "*") << endl;
    cout << "Test Case 3: " << isMatch("cb", "?a") << endl;
    cout << "Test Case 4: " << isMatch("aa", "a*") << endl;
    cout << "Test Case 5: " << isMatch("adceb", "*a*b") << endl;
    cout << "Test Case 6: " << isMatch("acdcb", "a*c?b") << endl;

    return 0;
}
```

Output:-

```
Test Case 1: false
Test Case 2: true
Test Case 3: false
Test Case 4: true
Test Case 5: true
Test Case 6: false


...Program finished with exit code 0
Press ENTER to exit console.
```

Q5. **Special Binary String**

Special binary strings are binary strings with the following two properties:

The number of 0's is equal to the number of 1's.

Every prefix of the binary string has at least as many 1's as 0's.

You are given a special binary string s.

A move consists of choosing two consecutive, non-empty, special substrings of s, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

Return the lexicographically largest resulting string possible after applying the mentioned operations on the string.

Example 1:

Input: s = "11011000"

Output: "11100100"

Explanation: The strings "10" [occuring at s[1]] and "1100" [at s[3]] are swapped.

This is the lexicographically largest string possible after some number of swaps.

Example 2:

Input: s = "10"

Output: "10"

Constraints:

1 <= s.length <= 50

s[i] is either '0' or '1'.

s is a special binary string.

Program Code:-

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

string makeLargestSpecial(string s) {
    if (s.length() <= 1) return s;

    vector<string> substrings;
    int count = 0, start = 0;

    for (int i = 0; i < s.length(); i++) {
```

```cpp
        count += (s[i] == '1') ? 1 : -1;

        if (count == 0) {
            string sub = s.substr(start + 1, i - start - 1);
            substrings.push_back(makeLargestSpecial(sub));
            start = i + 1;
        }
    }

    sort(substrings.rbegin(), substrings.rend());

    string result = "1";
    for (const string& sub : substrings) {
        result += sub;
    }
    result += "0";

    return result;
}

int main() {
    string s1 = "11011000";
    string s2 = "10";
    string s3 = "110100110";
    cout << "Largest Special Binary String for " << s3 << " : " <<
makeLargestSpecial(s3) << endl;
```

```
    return 0;

}
```

Output:-

```
Largest Special Binary String for 110100110 : 1100


...Program finished with exit code 0
Press ENTER to exit console.
```