Name: Nipun Chugh
UID: 22BCS10636
Class: KPIT - 901 / A

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 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).

Q2. Given a singly linked list, reverse the list, and return the reversed list.

Q3. 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.

Q4. 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).

Q5. 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.

Solutions :

A1. Fibonacci Series Using Recursion

```
#include <iostream>

using namespace std;

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

int main(int argc, char* argv[]) {
   int n = 0;
   cin >> n;

   for (int i = 0; i < n; i++) { cout << fibo(i) << endl; }
   return 0;
}
```

Output :

```
10
0 1 1 2 3 5 8 13 21 34
```

A2. Reverse Linked List

```
#include <iostream>

using namespace std;

struct Node{
   int   data;
   Node* next;
   Node(int data) : data(data), next(nullptr) {}
};

void i_reverse_list(Node& List) {
   Node* current = &List;
   Node* next    = nullptr;
   Node* prev    = nullptr;

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

void r_reverse_list(Node* List) {
   if (List->next == nullptr || List == nullptr) { return; }

   Node* rest = List->next;

   r_reverse_list(rest);

   List->next->next = List;
   List->next       = nullptr;
   List = rest;
}

int main(int argc, char* argv[]) {
   Node N1(1), N2(2), N3(3), N4(4);

   N1.next = &N2;
   N2.next = &N3;
   N3.next = &N4;
   N4.next = nullptr;

   Node* ptr = &N1;
   cout << "Original List            : ";
   while (ptr != nullptr) {
      cout << ptr->data << "->";
      ptr = ptr->next;
   }
   cout << "NULL" << endl;
   r_reverse_list(&N1);

   ptr = &N4;
   cout << "Recursively reversed List : ";
   while (ptr != nullptr) {
```

```
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL" << endl;

    i_reverse_list(N4);

    ptr = &N1;
    cout << "Iteratively reversed List : ";
    while (ptr != nullptr) {
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL" << endl;

    return 0;
}
```

Output :

```
Original List              : 1->2->3->4->NULL
Recursively reversed List : 4->3->2->1->NULL
Iteratively reversed List : 1->2->3->4->NULL
```

A3. Add Two Numbers

```
#include <iostream>
#include <cmath>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int data) : data(data), next(nullptr) {}
};

int list_to_num(Node* List) {
    int retval = 0;
    int k = 0;
    while (List != nullptr) {
        retval += List->data * pow(10, k);
        k++;
        List = List->next;
    }
    return retval;
}

Node* num_to_list(int num) {
    if (num == 0) { return new Node(0); }
    Node* retval = new Node(num % 10);
    num /= 10;
    Node* ptr = retval;

    while (num > 0) {
        Node* t = new Node(num % 10);
        ptr->next = t;
        ptr = t;
```

```
        num /= 10;
    }
    return retval;
}

int main(int argc, char* argv[]) {
    Node L1(2);
    Node L2(4);
    Node L3(3);

    L1.next = &L2;
    L2.next = &L3;

    Node M1(5);
    Node M2(6);
    Node M3(4);

    M1.next = &M2;
    M2.next = &M3;

    Node* ptr = &L1;
    while (ptr != nullptr) {
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL + ";

    ptr = &M1;
    while (ptr != nullptr) {
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL = ";

    ptr = num_to_list(list_to_num(&L1) + list_to_num(&M1));

    while (ptr != nullptr) {
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL" << endl;

    return 0;
}
```

Output :

```
2->4->3->NULL + 5->6->4->NULL = 7->0->8->NULL
```

A4. Wildcard Matching

```
#include <iostream>

using namespace std;

bool wildCard(string txt, string pat) {
```

```
    int n = txt.length();
    int m = pat.length();
    int i = 0, j = 0, startIndex = -1, match = 0;

    while (i < n) {
        if (j < m && (pat[j] == '?' || pat[j] == txt[i])) {
            i++;
            j++;
        } else if (j < m && pat[j] == '*') {
            startIndex = j;
            match = i;
            j++;
        } else if (startIndex != -1) {
            j = startIndex + 1;
            match++;
            i = match;
        } else { return false; }
    }
    while (j < m && pat[j] == '*') { j++; }
    return j == m;
}

int main() {
    string txt = "baaabab";
    string pat = "*****ba*****ab";

    cout << "String  : " << txt << endl;
    cout << "Pattern : " << pat << endl;
    cout << "Result  : " << (wildCard(txt, pat) ? "true" : "false");
}
```

Output :

```
String  : baaabab
Pattern : *****ba*****ab
Result  : true
```

A5. Special Binary String

```
#include <iostream>
#include <numeric>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

string makeLargestSpecial(string s) {
    if (s.empty()) return s;
    vector<string> specials;

    int counter  = 0;
    int startIdx = 0;

    for (int currentIdx = 0; currentIdx < s.size(); ++currentIdx) {
        counter += s[currentIdx] == '1' ? 1 : -1;
```

```
        if (counter == 0) {
            specials.push_back(
                "1" + makeLargestSpecial(
                    s.substr(
                        startIdx + 1,
                        currentIdx - startIdx - 1
                    )
                ) + "0"
            );
            startIdx = currentIdx + 1;
        }
    }

    sort(specials.begin(), specials.end(), greater<string>());

    return accumulate(specials.begin(), specials.end(), string{});
}

int main() {
    string s = "11011000";
    cout << "String : " << s << endl;
    cout << "Result : " << makeLargestSpecial(s);
}
```

Output :

```
String : 11011000
Result : 11100100
```