DOMAIN WINTER WINNING CAMP

Student Name: Unnati Srivastava **UID: 22BCS13475**

Branch: BE-CSE **Section/Group: TPP_FL_603-A**

DAY 3:

QUES 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.

Solution:

```
#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 the value of n: ";
    cin >> n;
    int result = fibonacci(n);
    cout << "F(" << n << ") = " << result << endl;
    return 0;
}
```

```
Enter the value of n: 5
F(5) = 5
```

QUES 2: Reverse Linked List

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

Solution:

```
#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* current = head;

    while (current) {
        ListNode* nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
```

```
    }
    cout << endl;
}
int main() {
    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);
    head = reverseList(head);
    cout << "Reversed list: ";
    printList(head);
    return 0;
}
```

```
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
```

QUES 3: 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.

Solution:

```
#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 dummy(0), *current = &dummy;
    int carry = 0;
    while (l1 || l2 || carry) {
        int sum = (l1 ? l1->val : 0) + (l2 ? l2->val : 0) + carry;
        carry = sum / 10;
        current = current->next = new ListNode(sum % 10);
        if (l1) l1 = l1->next;
        if (l2) l2 = l2->next;
    }
    return dummy.next;
}
void printList(ListNode* head) {
    while (head) cout << head->val << " ", head = head->next;
    cout << endl;
}
int main() {
    ListNode* l1 = new ListNode(2);
    l1->next = new ListNode(4);
    l1->next->next = new ListNode(3);
    ListNode* l2 = new ListNode(5);
    l2->next = new ListNode(6);
    l2->next->next = new ListNode(4);
```

```
    printList(addTwoNumbers(l1, l2));
    return 0;
}
```

```
7 0 8
```

QUES 4: 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).

Solution:

```
#include <iostream>
#include <vector>
using namespace std;
bool isMatch(string s, string p) {
    int m = s.size(), n = p.size();
    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() {
    string s = "aa", p = "a";
    cout << (isMatch(s, p) ? "true" : "false") << endl;
    return 0;
}
```

```
false
```

QUES 5: 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.

Solution:

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

```
string makeLargestSpecial(string s) {
    vector<string> specials;
    int count = 0, start = 0;
    for (int i = 0; i < s.size(); ++i) {
        count += (s[i] == '1') ? 1 : -1;
        if (count == 0) {
            specials.push_back("1" + makeLargestSpecial(s.substr(start + 1, i - start - 1)) + "0");
            start = i + 1;
        }
    }
    sort(specials.rbegin(), specials.rend());
    string result;
    for (const string& sp : specials) result += sp;
    return result;
}
int main() {
    string s = "11011000";
    cout << makeLargestSpecial(s) << endl;
    return 0;
}
```

```
11100100
```

Q:1 Write a Function to print first name and last name using function

You are given the firstname and lastname of a person on two different lines. Your task is to read them and print the following using function:

Hello firstname lastname! You just delved into function.

Code:

```
#include <iostream>
#include <string>

using namespace std;

// Function to print the full name
void print_full_name(string first, string last) {
    cout << "Hello " << first << " " << last << "! You just delved into function." << endl;
}
int main() {
    string first_name, last_name;
    // Input first name and last name
    cout << "Enter first name: ";
    cin >> first_name;
    cout << "Enter last name: ";
    cin >> last_name;
    // Call the function to print full name
    print_full_name(first_name, last_name);
    return 0;
}
```

```
Enter first name: Narendra
Enter last name: Modi
Hello Narendra Modi! You just delved into function.
```

Q:2 Find GCD of Number Using Function

Given an integer array nums, return the greatest common divisor of the smallest number and largest number in nums.

The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.

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

using namespace std;

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int findGCD(vector<int>& nums) {
    int minNum = *min_element(nums.begin(), nums.end());
    int maxNum = *max_element(nums.begin(), nums.end());
    return gcd(minNum, maxNum);
}

int main() {
    vector<int> nums = {2, 5, 6, 9, 10};
    cout << "GCD: " << findGCD(nums) << endl;
```

```
    return 0;
}
```

```
GCD: 2
```

Q3 : Longest Substring Without Repeating Characters

You are working on building a text editor application. One of the features you're focusing on is a "word suggestion" system. The editor needs to identify the longest sequence of characters typed without repeating any letters to suggest potential words or phrases. To accomplish this, you must efficiently find the length of the longest substring of unique characters as the user types.

```
#include <iostream>
#include <string>
#include <unordered_set>

using namespace std;

int lengthOfLongestSubstring(string s) {
    unordered_set<char> seen;
    int maxLength = 0, left = 0;

    for (int right = 0; right < s.length(); ++right) {
        while (seen.find(s[right]) != seen.end()) {
            seen.erase(s[left]);
            ++left;
        }
        seen.insert(s[right]);
        maxLength = max(maxLength, right - left + 1);
    }
```

```
    return maxLength;
}

int main() {
    string input;
    cout << "Enter a string: ";
    cin >> input;
    cout << "Length of longest substring without repeating characters: " <<
lengthOfLongestSubstring(input) << endl;
    return 0;
}
```

```
Enter a string: Hello
Length of longest substring without repeating characters: 3
```

Q4 : Maximum Subarray Product

You are developing a financial analysis application where users input daily stock price changes in an array. Your task is to determine the maximum profit or loss achievable over a contiguous period, considering that the profit/loss for a period is calculated by multiplying the daily percentage changes. This problem is critical for finding optimal periods to make decisions for investments or short-term trading.

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

using namespace std;

double maxProduct(vector<double>& changes) {
```

```
    double maxProd = changes[0];
    double minProd = changes[0];
    double result = changes[0];

    for (size_t i = 1; i < changes.size(); ++i) {
        if (changes[i] < 0) {
            swap(maxProd, minProd);
        }
        maxProd = max(changes[i], maxProd * changes[i]);
        minProd = min(changes[i], minProd * changes[i]);

        result = max(result, maxProd);
    }

    return result;
}

int main() {
    vector<double> changes;
    int n;

    cout << "Enter the number of daily percentage changes: ";
    cin >> n;

    cout << "Enter the daily percentage changes: ";
    for (int i = 0; i < n; ++i) {
        double change;
        cin >> change;
```

```
        changes.push_back(change);
    }

    cout << "Maximum subarray product: " << maxProduct(changes) << endl;

    return 0;
}
```

```
Enter the number of daily percentage changes: 5
Enter the daily percentage changes: 1.2 -0.5 -2 3 0.1
Maximum subarray product: 3.6
```

Q5 - There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.

Children with a higher rating get more candies than their neighbors.

Return the minimum number of candies you need to have to distribute the candies to the children.

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

using namespace std;

int candy(vector<int>& ratings) {
    int n = ratings.size();
    vector<int> candies(n, 1);
```

```
    for (int i = 1; i < n; ++i) {
        if (ratings[i] > ratings[i - 1]) {
            candies[i] = candies[i - 1] + 1;
        }
    }

    for (int i = n - 2; i >= 0; --i) {
        if (ratings[i] > ratings[i + 1]) {
            candies[i] = max(candies[i], candies[i + 1] + 1);
        }
    }

    return accumulate(candies.begin(), candies.end(), 0);
}

int main() {
    vector<int> ratings;
    int n;

    cout << "Enter the number of children: ";
    cin >> n;

    cout << "Enter the ratings of the children: ";
    for (int i = 0; i < n; ++i) {
        int rating;
        cin >> rating;
        ratings.push_back(rating);
    }
```

```
    cout << "Minimum candies required: " << candy(ratings) << endl;

    return 0;
}
```

```
Enter the number of children: 5
Enter the ratings of the children: 1 1 0 2 1
Minimum candies required: 7
```