DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Gurnoor Oberoi **UID: 22BCS15716**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-3 [21-12-2024]

1. Fibonacci Series Using Recursion *(Very Easy)*

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).

Implementation/Code:

```
#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;

    if (n < 0) {
```

```
        cout << "Fibonacci sequence is not defined for negative numbers." << endl;
    } else {
        cout << "Fibonacci number F(" << n << ") = " << fibonacci(n) << endl;
    }
    return 0;
}
```

Output:

```
Enter the value of n: 2
Fibonacci number F(2) = 1
```

2. Reverse Linked List *(Easy)*

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

Implementation/Code:

```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int value) : val(value), next(nullptr) {}
};
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* current = head;
    ListNode* next = nullptr;
    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void printList(ListNode* head) {
    ListNode* current = head;
    while (current != nullptr) {
```

```
        cout << current->val << " ";
        current = current->next;
    }
    cout << endl;
}
ListNode* createListFromUserInput(int n) {
    if (n == 0) return nullptr;
    cout << "Enter the values of the list: ";
    int value;
    cin >> value;
    ListNode* head = new ListNode(value);
    ListNode* current = head;
    for (int i = 1; i < n; ++i) {
        cin >> value;
        current->next = new ListNode(value);
        current = current->next;
    }
    return head;
}

int main() {
    int n;
    cout << "Enter the number of elements in the list: ";
    cin >> n;
    if (n <= 0) {
        cout << "The list must have at least one element." << endl;
        return 0;
    }
    ListNode* head = createListFromUserInput(n);
    cout << "Original list: ";
    printList(head);
    head = reverseList(head);
    cout << "Reversed list: ";
    printList(head);
    return 0;
}
```

Output:

```
Enter the number of elements in the list: 5
Enter the values of the list: 1 2 3 4 5
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
```

3. Add two Numbers *(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.

Implementation/Code:

```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;

    ListNode(int value) : val(value), 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;
```

```
        current->next = new ListNode(sum % 10);
        current = current->next;
    }
    return dummyHead->next;
}
ListNode* createListFromUserInput(int n) {
    if (n == 0) return nullptr;
    cout << "Enter the values of the list: ";
    int value;
    cin >> value;
    ListNode* head = new ListNode(value);
    ListNode* current = head;
    for (int i = 1; i < n; ++i) {
        cin >> value;
        current->next = new ListNode(value);
        current = current->next;
    }
    return head;
}
void printList(ListNode* head) {
    ListNode* current = head;
    while (current != nullptr) {
        cout << current->val << " ";
        current = current->next;
    }
    cout << endl;
}
int main() {
    int n1, n2;
    cout << "Enter the number of digits in the first number: ";
    cin >> n1;
    ListNode* l1 = createListFromUserInput(n1);
    cout << "Enter the number of digits in the second number: ";
    cin >> n2;
    ListNode* l2 = createListFromUserInput(n2);
    ListNode* result = addTwoNumbers(l1, l2);
    cout << "Resultant list: ";
    printList(result);
```

```
    return 0;
}
```

Output:

```
Enter the values of the list: 2 4 3
Enter the number of digits in the second number: 3
Enter the values of the list: 5 6 4
Resultant list: 7 0 8
```

4. WildCard Matching *(Hard)*

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).

Implementation/Code:

```
#include <iostream>
#include <vector>
using namespace std;
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() {
    string s, p;
    cout << "Enter the input string: ";
    cin >> s;

    cout << "Enter the pattern: ";
    cin >> p;
    if (isMatch(s, p)) {
        cout << "The string matches the pattern." << endl;
    } else {
        cout << "The string does not match the pattern." << endl;
    }
    return 0;
}
```

Output:

```
Enter the input string: aa
Enter the pattern: a
The string does not match the pattern.
```

5. Special Binary String *(Very Hard)*

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.

Implementation/Code:

```
#include <iostream>
```

```
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
string makeLargestSpecial(string s) {
    vector<string> specialSubstrings;
    int count = 0, start = 0;
    for (int i = 0; i < s.size(); ++i) {
        count += (s[i] == '1' ? 1 : -1);
        if (count == 0) {
            string inner = makeLargestSpecial(s.substr(start + 1, i - start - 1));
            specialSubstrings.push_back("1" + inner + "0");
            start = i + 1;
        }
    }
    sort(specialSubstrings.begin(), specialSubstrings.end(), greater<string>());
    string result;
    for (const string& str : specialSubstrings) {
        result += str;
    }
    return result;
}
int main() {
    string s;
    cout << "Enter a special binary string: ";
    cin >> s;
    string largestSpecial = makeLargestSpecial(s);
    cout << "Lexicographically largest special binary string: " << largestSpecial << endl;
    return 0;
}
```

Output:

```
Enter a special binary string: 11011000
Lexicographically largest special binary string: 11100100
```

6. **Write a Function to print first name and last name using function** ***(Very Easy)***

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.

Implementation/Code:

```
#include <iostream>
#include <string>
using namespace std;
void printGreeting(const string& firstName, const string& lastName) {
   cout << "Hello " << firstName << " " << lastName << "! You just delved into function." << endl;
}
int main() {
   string firstName, lastName;
   cout << "Enter first name: ";
   getline(cin, firstName);
   cout << "Enter last name: ";
   getline(cin, lastName);
   printGreeting(firstName, lastName);

   return 0;
}
```

Output:

```
Enter first name: Ross
Enter last name: Taylor
Hello Ross Taylor! You just delved into function.
```

7. Find GCD of Number Using Function *(Easy)*

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.

Implementation/Code:

```
#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 smallest = *min_element(nums.begin(), nums.end());
    int largest = *max_element(nums.begin(), nums.end());
    return gcd(smallest, largest);
}

int main() {
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
    int result = findGCD(nums);
    cout << "The GCD of the smallest and largest numbers is: " << result << endl;
    return 0;
}
```

Output:

```
Enter the number of elements in the array: 5
Enter the elements of the array: 2 5 6 9 10
The GCD of the smallest and largest numbers is: 2
```

8. Longest Substring without Repeating Characters *(Medium)*

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.

Description:

Write a function that takes a string as input and returns the length of the longest substring without repeating characters. A substring is a contiguous sequence of characters within the string.

Implementation/Code:

```
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;
int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> charIndex;
    int maxLength = 0;
    int start = 0;
    for (int end = 0; end < s.length(); ++end) {
        char currentChar = s[end];
        if (charIndex.find(currentChar) != charIndex.end() && charIndex[currentChar] >=
start) {
            start = charIndex[currentChar] + 1;
        }
        charIndex[currentChar] = end;
        maxLength = max(maxLength, end - start + 1);
    }
    return maxLength;
}
int main() {
    string input;
    cout << "Enter a string: ";
    getline(cin, input);
    int result = lengthOfLongestSubstring(input);
    cout << "The length of the longest substring without repeating characters is: " << result
<< endl;
    return 0;
}
```

Output:

```
Enter a string: abcabcbb
The length of the longest substring without repeating characters is: 3
```

9. Maximum Subarray Product *(Hard)*

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.

Description:

Write a function that takes an integer array as input and returns the maximum product of a contiguous subarray. The array can contain both positive and negative integers, and your function must account for these to find the optimal subarray.

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int maxProduct(vector<int>& nums) {
    if (nums.empty()) return 0;
    int maxProduct = nums[0];
    int currentMax = nums[0];
    int currentMin = nums[0];
    for (int i = 1; i < nums.size(); ++i) {
        int tempMax = currentMax;
        currentMax = max({nums[i], currentMax * nums[i], currentMin * nums[i]});
        currentMin = min({nums[i], tempMax * nums[i], currentMin * nums[i]});
        maxProduct = max(maxProduct, currentMax);
    }
    return maxProduct;
}
int main() {
    int n;
    cout << "Enter the number of daily stock price changes: ";
    cin >> n;
    vector<int> changes(n);
```

```
    cout << "Enter the daily stock price changes: ";
    for (int i = 0; i < n; ++i) {
        cin >> changes[i];
    }
    int result = maxProduct(changes);
    cout << "The maximum product of a contiguous subarray is: " << result << endl;

    return 0;
}
```

Output:

```
Enter the number of daily stock price changes: 4
Enter the daily stock price changes: 2 3 -2 4
The maximum product of a contiguous subarray is: 6
```

10. Minimum Candies Distribution *(Very Hard)*

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int candy(vector<int>& ratings) {
    int n = ratings.size();
    if (n == 0) return 0;
    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);
        }
    }
    int totalCandies = 0;
    for (int candy : candies) {
        totalCandies += candy;
    }
    return totalCandies;
}
int main() {
    int n;
    cout << "Enter the number of children: ";
    cin >> n;
    vector<int> ratings(n);
    cout << "Enter the ratings of the children: ";
    for (int i = 0; i < n; ++i) {
        cin >> ratings[i];
    }
    int result = candy(ratings);
    cout << "The minimum number of candies required is: " << result << endl;
    return 0;
}
```

Output:

```
Enter the number of children: 3
Enter the ratings of the children: 1 0 2
The minimum number of candies required is: 5
```