DOMAIN WINTER CAMP WORKSHEET
DAY-3

Student Name: Kethrin Naharwal **UID:** 22BCS12653
Branch: BE-CSE **Section:** 22_BCS_FL_IOT_603

Very Easy

1Q .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

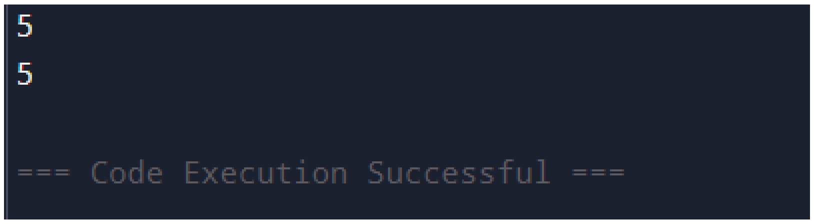
F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

SOLUTION :-

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

int main() {
    int n;
    cin >> n;
    if (n == 0) {
        cout << 0;
        return 0;
    }
    if (n == 1) {
        cout << 1;
        return 0;
    }
    int a = 0, b = 1, c;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    cout << c;
    return 0;
}
```

OUTPUT:



```
5
5

=== Code Execution Successful ===
```

Easy

2Q. Reverse Linked List

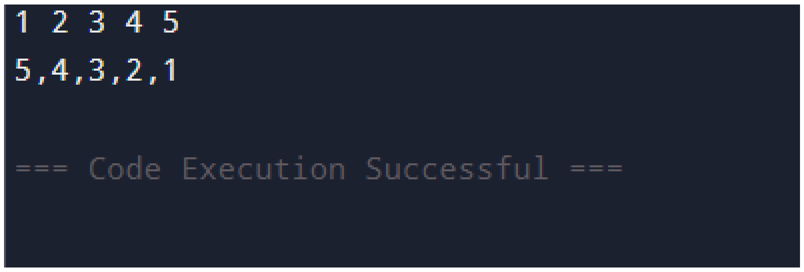
Given the head of a singly linked list, reverse the list, and return the reversed list.

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* curr = head;
    while (curr) {
        ListNode* nextTemp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}

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

int main() {
    int n, val;
    cin >> n;
    ListNode* head = nullptr;
    ListNode* tail = nullptr;
    for (int i = 0; i < n; ++i) {
        cin >> val;
        ListNode* newNode = new ListNode(val);
        if (!head) head = tail = newNode;
        else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    head = reverseList(head);
    printList(head);
    return 0;
}
```

OUTPUT:



```
1 2 3 4 5
5,4,3,2,1

=== Code Execution Successful ===
```

Medium

3 Q. Add Two Numbers

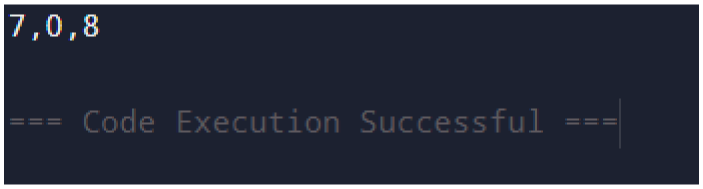
You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

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 = new ListNode(0);
    ListNode* current = dummy;
    int carry = 0;
    while (l1 || l2 || carry) {
        int sum = (l1 ? l1->val : 0) + (l2 ? l2->val : 0) + carry;
        carry = sum / 10;
        current->next = new ListNode(sum % 10);
        current = current->next;
        if (l1) l1 = l1->next;
        if (l2) l2 = l2->next;
    }
    return dummy->next;
}

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

ListNode* createList(const int arr[], int n) {
    ListNode* head = nullptr, *tail = nullptr;
    for (int i = 0; i < n; ++i) {
        ListNode* newNode = new ListNode(arr[i]);
        if (!head) head = tail = newNode;
        else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

int main() {
    int l1_vals[] = {2, 4, 3};
    int l2_vals[] = {5, 6, 4};

    ListNode* l1 = createList(l1_vals, 3);
    ListNode* l2 = createList(l2_vals, 3);
    ListNode* result = addTwoNumbers(l1, l2);
    printList(result);

    return 0;
}
```

OUTPUT:


```
7,0,8

=== Code Execution Successful ===
```

Hard

4 Q. Wildcard Matching

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

'?' Matches any single character.

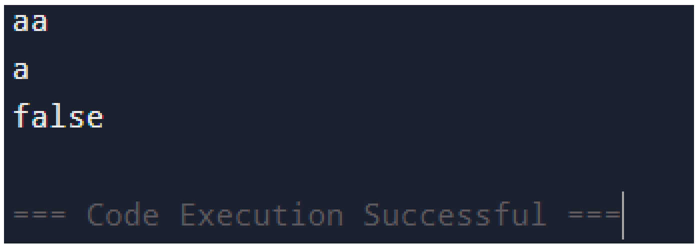
'*' Matches any sequence of characters (including the empty sequence).

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][j - 1] || dp[i - 1][j];
           }
      }
   }

   return dp[m][n];
}

int main() {
   string s, p;
   cin >> s >> p;
   cout << (isMatch(s, p) ? "true" : "false");
   return 0;
}
```

OUTPUT:



```
aa
a
false

=== Code Execution Successful ===
```

Very Hard

5 Q. 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.

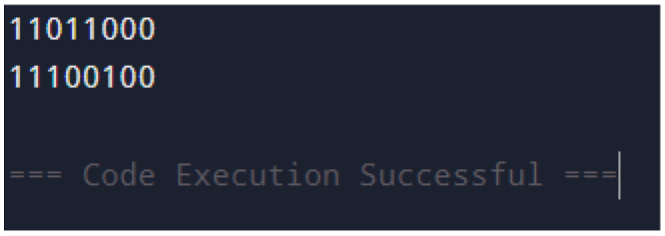
A move consists of choosing two consecutive, non-empty, special substrings of s, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

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

SOLUTION :-

```
#include <iostream>
#include <vector>
#include <string>
#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) {
            string substring = "1" + makeLargestSpecial(s.substr(start + 1, i - start - 1)) + "0";
            specials.push_back(substring);
            start = i + 1;
        }
    }

    sort(specials.rbegin(), specials.rend());
    string result;
    for (const string& sub : specials) result += sub;

    return result;
}

int main() {
    string s;
    cin >> s;
    cout << makeLargestSpecial(s);
    return 0;
}
```

OUTPUT:



```
11011000
11100100

=== Code Execution Successful ===
```

Function Questions

Very Easy

1 Q. 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.

Function Description:

Complete the print_full_name function in the editor below.

print_full_name has the following parameters:

string first: the first name

string last: the last name

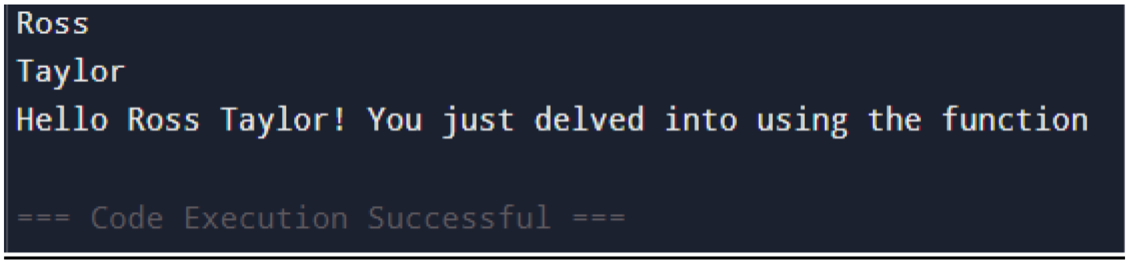
Prints

string: 'Hello firstname lastname ! You just delved into using the function' where firstname and lastname are replaced with first and last.

SOLUTION :-

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

void print_full_name(string first, string last) {
    cout << "Hello " << first << " " << last << "! You just delved into using the function";
}

int main() {
    string first, last;
    cin >> first >> last;
    print_full_name(first, last);
    return 0;
}
```

OUTPUT:



```
Ross
Taylor
Hello Ross Taylor! You just delved into using the function

=== Code Execution Successful ===
```

Easy

2 Q 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.

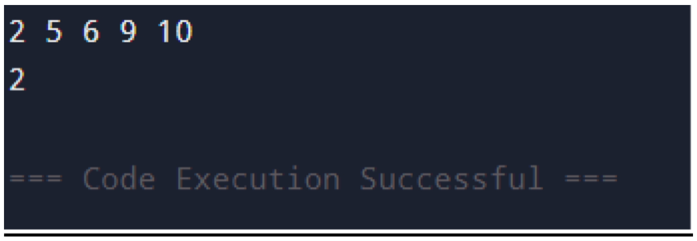
The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.

SOLUTION :

```
#include <iostream>
```

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

int gcd(int a, int b) {
   return b == 0 ? a : gcd(b, a % b);
}

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() {
   int n;
   cin >> n;
   vector<int> nums(n);
   for (int i = 0; i < n; ++i) {
      cin >> nums[i];
   }
   cout << findGCD(nums);
   return 0;
}
```

OUTPUT:



```
2 5 6 9 10
2


=== Code Execution Successful ===
```

Medium

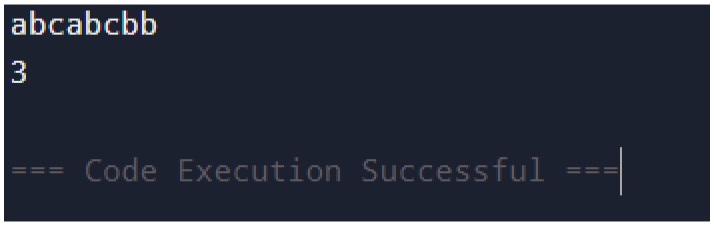
Q3 .: Longest Substring Without Repeating Characters

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.

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.

SOLUTION :
```
#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.size(); ++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;
   cin >> input;
   cout << lengthOfLongestSubstring(input);
   return 0;
}
```

OUTPUT:



```
abcabcbb
3

=== Code Execution Successful ===
```

Hard

Q4 : Maximum Subarray Product

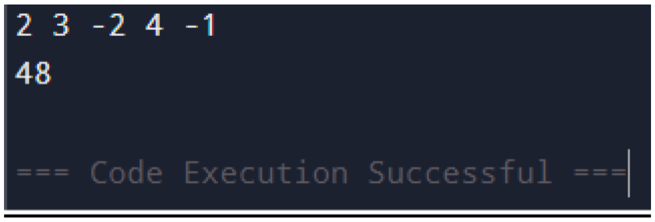
You are developing a financial analysis application where users input daily stock price changes in an array. Your task is to determine the maximum profit or loss achievable over a contiguous period, considering that the profit/loss for a period is calculated by multiplying the daily percentage changes.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

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.

SOLUTION :

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

int maxProduct(vector<int>& nums) {
    int n = nums.size();
    int maxProd = nums[0], minProd = nums[0], result = nums[0];

    for (int i = 1; i < n; ++i) {
        if (nums[i] < 0) {
            swap(maxProd, minProd);
        }
        maxProd = max(nums[i], maxProd * nums[i]);

        minProd = min(nums[i], minProd * nums[i]);
        result = max(result, maxProd);
    }

    return result;
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
    cout << maxProduct(nums);
    return 0;
}
```

OUTPUT:



```
2 3 -2 4 -1
48

=== Code Execution Successful ===
```

Very Hard

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.

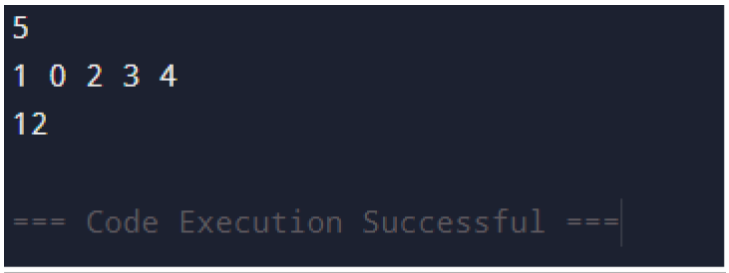
Children with a higher rating get more candies than their neighbors.

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

SOLUTION :

```
#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);
        }
    }

    int totalCandies = 0;
    for (int candyCount : candies) {
        totalCandies += candyCount;
    }
    return totalCandies;
}

int main() {
    int n;
    cin >> n;
    vector<int> ratings(n);
    for (int i = 0; i < n; ++i) {
        cin >> ratings[i];
    }
    cout << candy(ratings);
    return 0;
}
```

OUTPUT:



```
5
1 0 2 3 4
12

=== Code Execution Successful ===
```