Day 3

Function $ RECURSION DSA QUESTION

Very Easy

Q1 .Fibonnacci 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

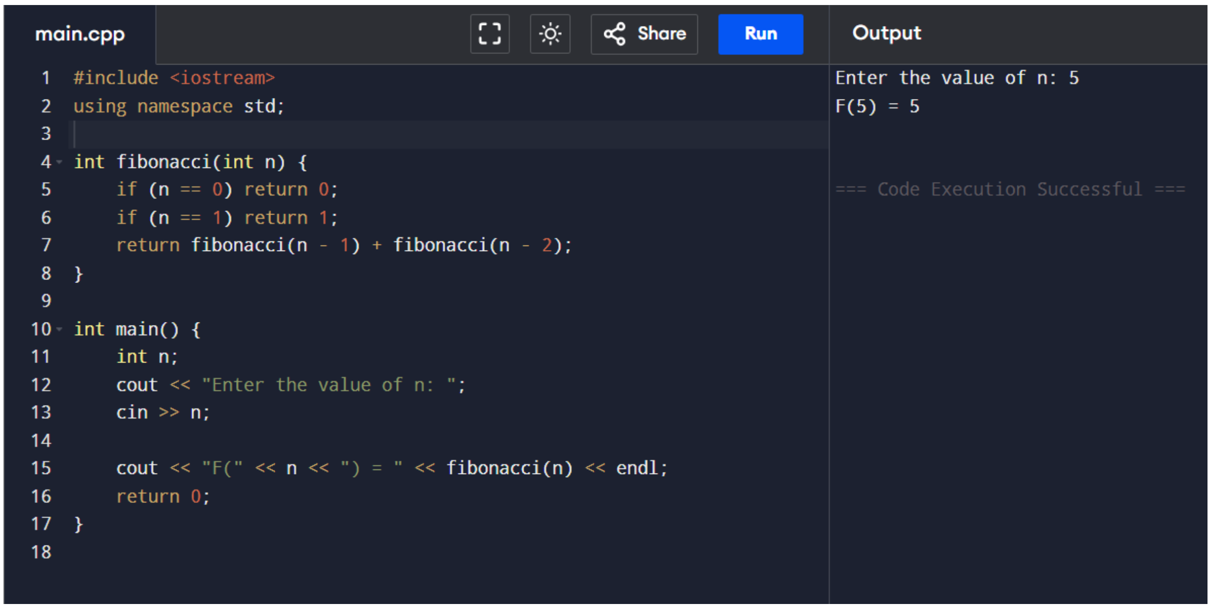
Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1

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;
  cout << "F(" << n << ") = " << fibonacci(n) << endl;
    return 0;
}
```

Output:



```
#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;

    cout << "F(" << n << ") = " << fibonacci(n) << endl;
    return 0;
}
```

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


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

Q2 . Factorial Of Number Using Recursion

Write a program that returns the value of N! (N factorial) using recursion.

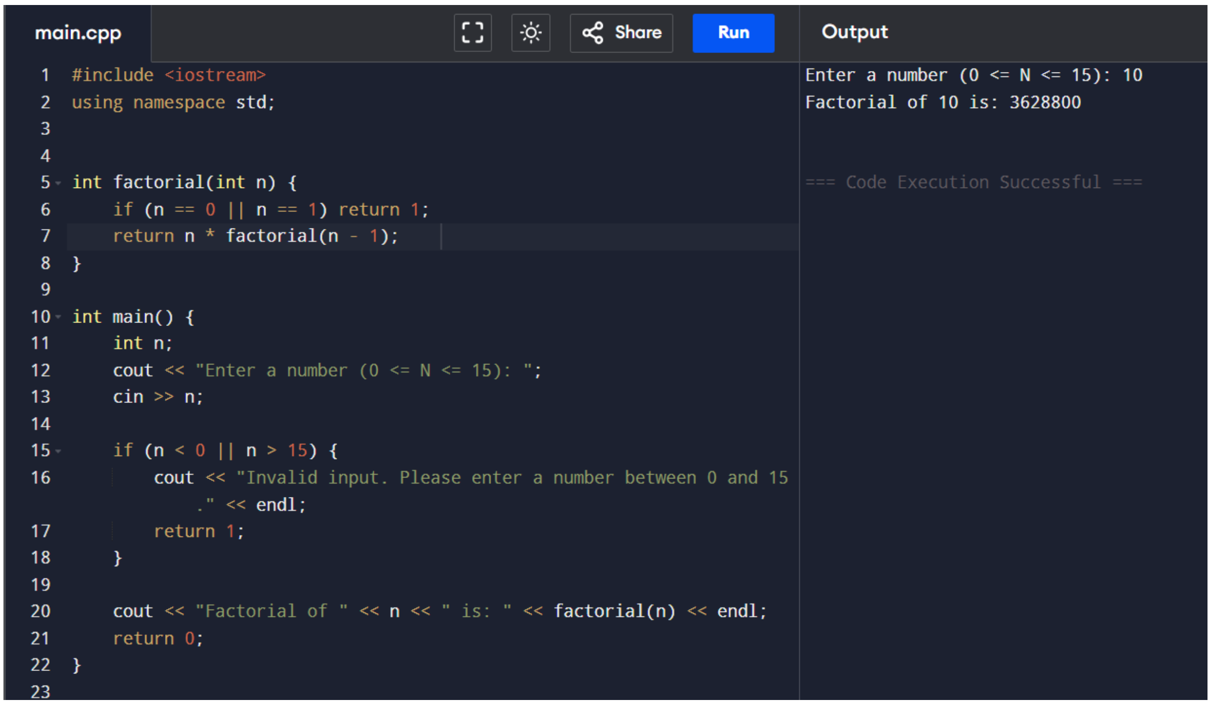
Note that N! =- 1*2*...*N

Also, 0! = 1 and 1! = 1.

Code:

```
#include <iostream>
using namespace std;
int factorial(int n) {
    if (n == 0 || n == 1) return 1;
    return n * factorial(n - 1);
}
int main() {
    int n;
    cout << "Enter a number (0 <= N <= 15): ";
    cin >> n;
  if (n < 0 || n > 15) {
        cout << "Invalid input. Please enter a number between 0 and 15." << endl;
        return 1;
    }
  cout << "Factorial of " << n << " is: " << factorial(n) << endl;
    return 0;
}
```

Output:



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


int factorial(int n) {
    if (n == 0 || n == 1) return 1;
    return n * factorial(n - 1);
}

int main() {
    int n;
    cout << "Enter a number (0 <= N <= 15): ";
    cin >> n;

    if (n < 0 || n > 15) {
        cout << "Invalid input. Please enter a number between 0 and 15
            ." << endl;
        return 1;
    }

    cout << "Factorial of " << n << " is: " << factorial(n) << endl;
    return 0;
}
```

```
Enter a number (0 <= N <= 15): 10
Factorial of 10 is: 3628800


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

Q3.Sum of Natural Number Using Recursion

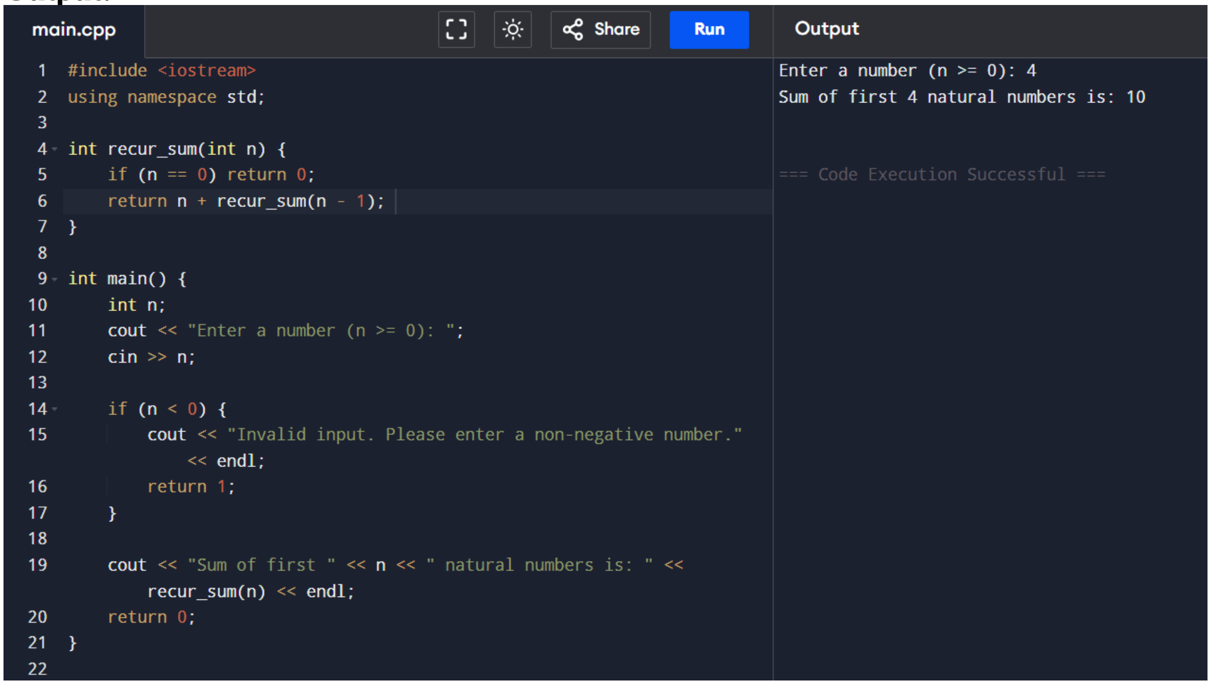
Given a number n, find sum of first n natural numbers. To calculate the sum, we will use a recursive function recur_sum().

Code:

```
#include <iostream>

using namespace std;
```

```
int recur_sum(int n) {
    if (n == 0) return 0;
    return n + recur_sum(n - 1);
}
int main() {
    int n;
    cout << "Enter a number (n >= 0): ";
    cin >> n;
if (n < 0) {
        cout << "Invalid input. Please enter a non-negative number." << endl;
        return 1;
    }
cout << "Sum of first " << n << " natural numbers is: " << recur_sum(n) << endl;
    return 0;
}
```

Output:



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

int recur_sum(int n) {
    if (n == 0) return 0;
    return n + recur_sum(n - 1);
}

int main() {
    int n;
    cout << "Enter a number (n >= 0): ";
    cin >> n;

    if (n < 0) {
        cout << "Invalid input. Please enter a non-negative number."
            << endl;
        return 1;
    }

    cout << "Sum of first " << n << " natural numbers is: " <<
        recur_sum(n) << endl;
    return 0;
}
```

```
Enter a number (n >= 0): 4
Sum of first 4 natural numbers is: 10


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

Easy

1. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:

Input: list1 = [1,2,4], list2 = [1,3,4]

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

Code:

```
#include <iostream>

using namespace std;

struct ListNode {

    int val;

    ListNode* next;

    ListNode() : val(0), next(nullptr) {}

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

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

};

ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

    if (!list1) return list2;

    if (!list2) return list1;

if (list1->val <= list2->val) {

        list1->next = mergeTwoLists(list1->next, list2);

        return list1;

    } else {

        list2->next = mergeTwoLists(list1, list2->next);
        return list2;
    }
}
ListNode* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val;
        if (head->next) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}
int main() {
    int arr1[] = {1, 2, 4};
    int arr2[] = {1, 3, 4};

    ListNode* list1 = createList(arr1, 3);
 ListNode* list2 = createList(arr2, 3);
cout << "List 1: ";
```

printList(list1);

cout << "List 2: ";

printList(list2);

ListNode* mergedList = mergeTwoLists(list1, list2);

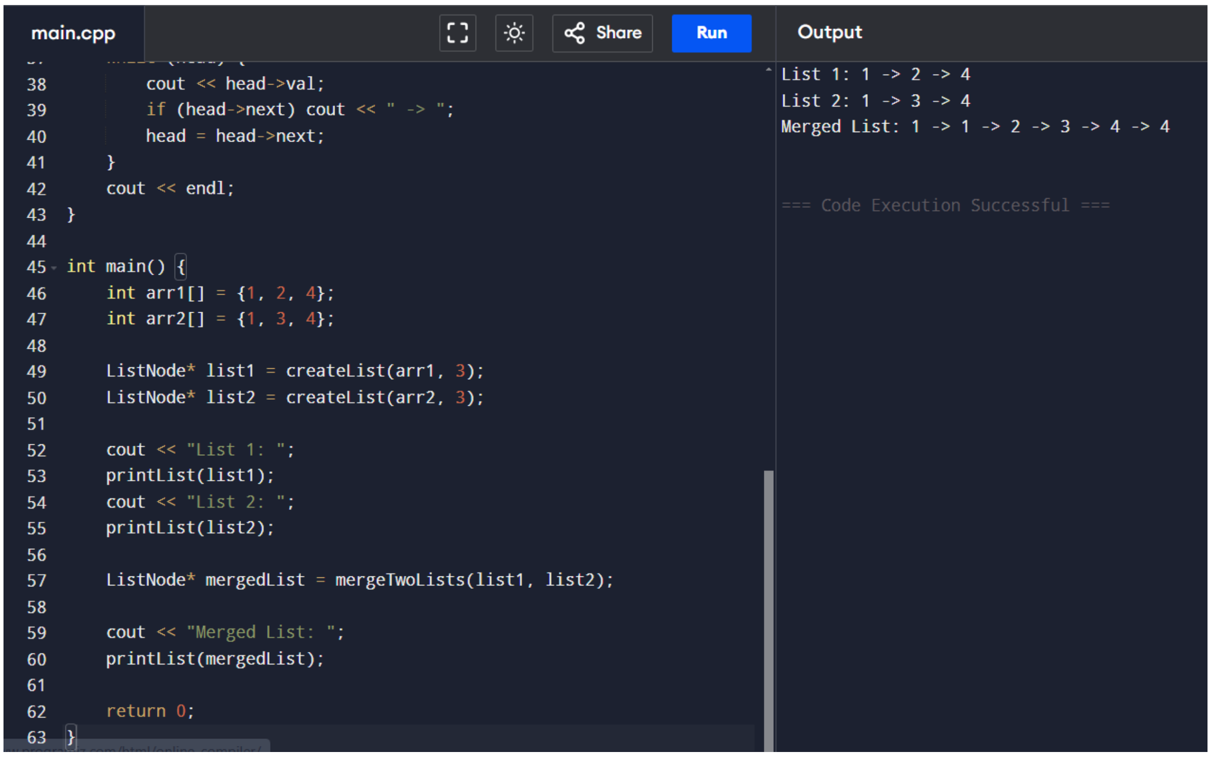
cout << "Merged List: ";

printList(mergedList);

return 0;

}

Output:



```
        cout << head->val;
        if (head->next) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    int arr1[] = {1, 2, 4};
    int arr2[] = {1, 3, 4};

    ListNode* list1 = createList(arr1, 3);
    ListNode* list2 = createList(arr2, 3);

    cout << "List 1: ";
    printList(list1);
    cout << "List 2: ";
    printList(list2);

    ListNode* mergedList = mergeTwoLists(list1, list2);

    cout << "Merged List: ";
    printList(mergedList);

    return 0;
}
```

```
List 1: 1 -> 2 -> 4
List 2: 1 -> 3 -> 4
Merged List: 1 -> 1 -> 2 -> 3 -> 4 -> 4


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

2. Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.

Example 1:

Input: head = [1,2,6,3,4,5,6], val = 6

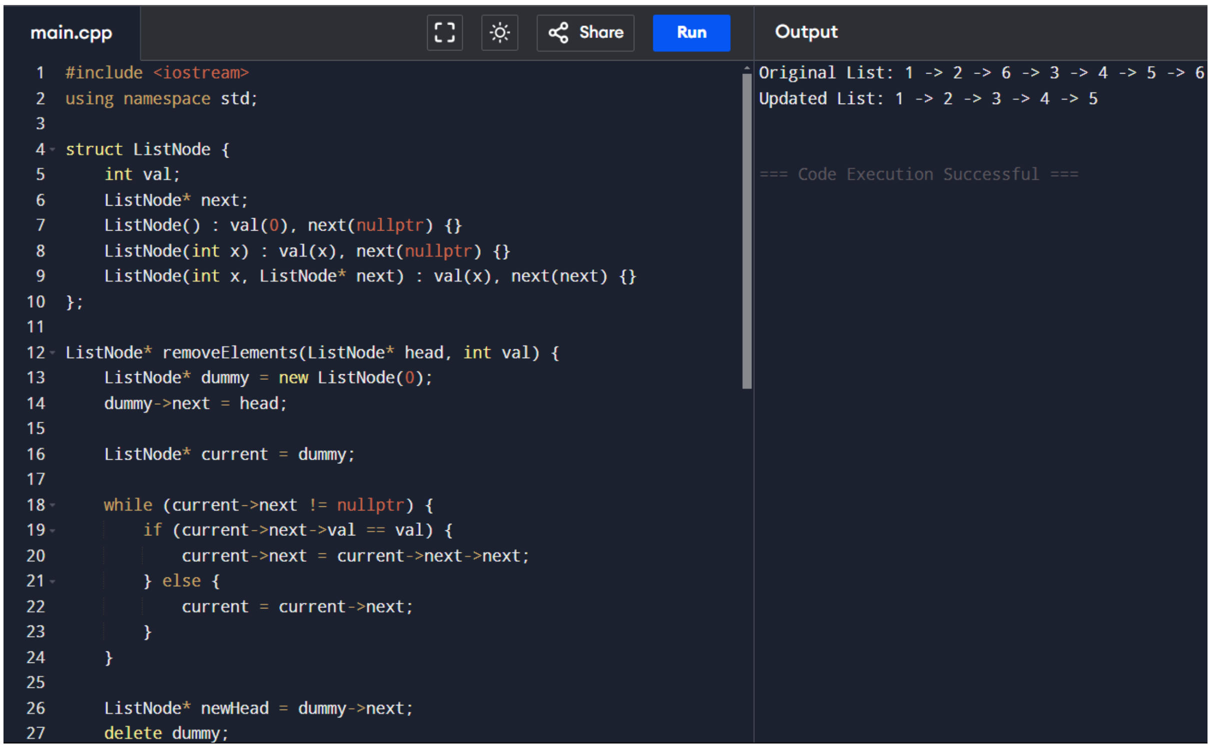
Output: [1,2,3,4,5]

Code:

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* removeElements(ListNode* head, int val) {
    ListNode* dummy = new ListNode(0);
    dummy->next = head;

    ListNode* current = dummy;

    while (current->next != nullptr) {
        if (current->next->val == val) {
            current->next = current->next->next;
        } else {
            current = current->next;
        }
    }

    ListNode* newHead = dummy->next;
    delete dummy;
    return newHead;
}
```

```
ListNode* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}

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

int main() {
    int arr[] = {1, 2, 6, 3, 4, 5, 6};
    int val = 6;

    ListNode* head = createList(arr, 7);

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

    ListNode* updatedList = removeElements(head, val);

    cout << "Updated List: ";
    printList(updatedList);

    return 0;
}
```

Output:



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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* removeElements(ListNode* head, int val) {
    ListNode* dummy = new ListNode(0);
    dummy->next = head;

    ListNode* current = dummy;

    while (current->next != nullptr) {
        if (current->next->val == val) {
            current->next = current->next->next;
        } else {
            current = current->next;
        }
    }

    ListNode* newHead = dummy->next;
    delete dummy;
```

```
Original List: 1 -> 2 -> 6 -> 3 -> 4 -> 5 -> 6
Updated List: 1 -> 2 -> 3 -> 4 -> 5


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

3. Reverse Linked List

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

Example 1:

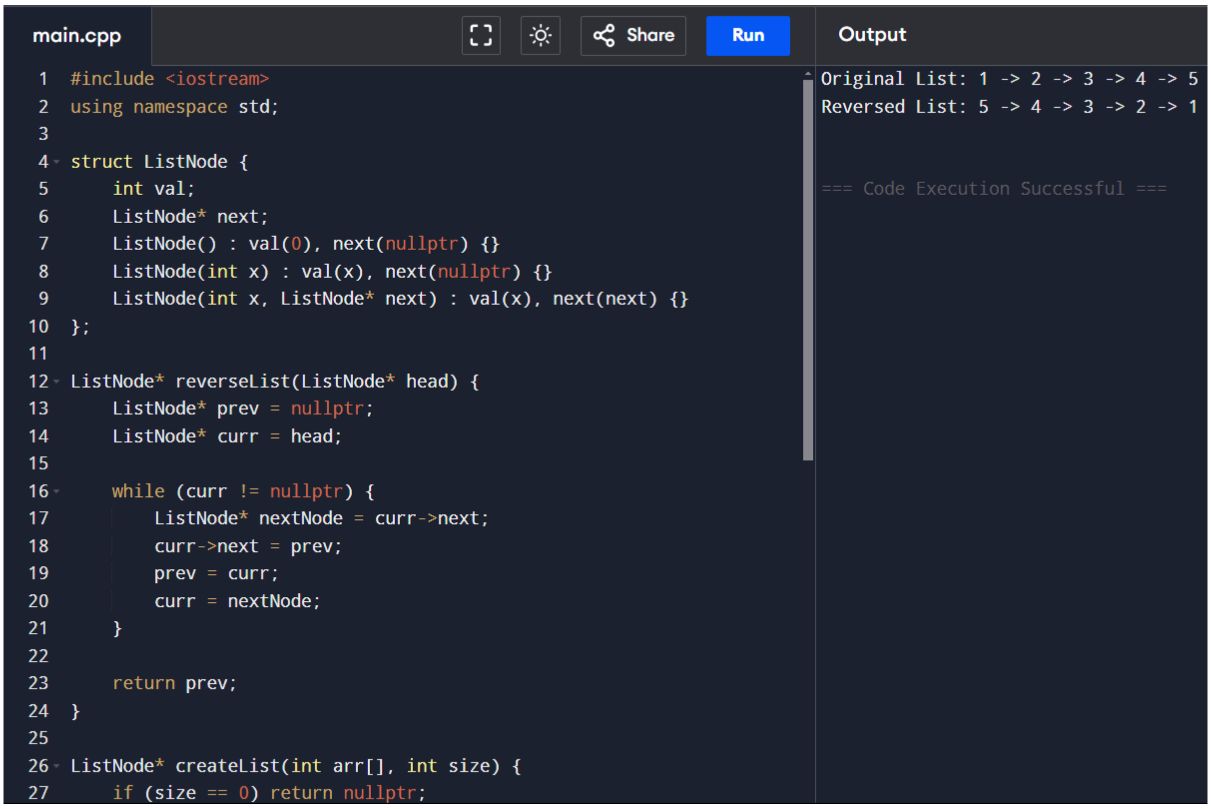
Input: head = [1,2,3,4,5]

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

Code:

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;

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

    return prev;
}

ListNode* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}

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

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    ListNode* head = createList(arr, 5);

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

    ListNode* reversedList = reverseList(head);

    cout << "Reversed List: ";
    printList(reversedList);

    return 0;
}
```

Output:



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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;

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

    return prev;
}

ListNode* createList(int arr[], int size) {
    if (size == 0) return nullptr;
```

```
Original List: 1 -> 2 -> 3 -> 4 -> 5
Reversed List: 5 -> 4 -> 3 -> 2 -> 1


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

Medium

Q1 . 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.

Code:

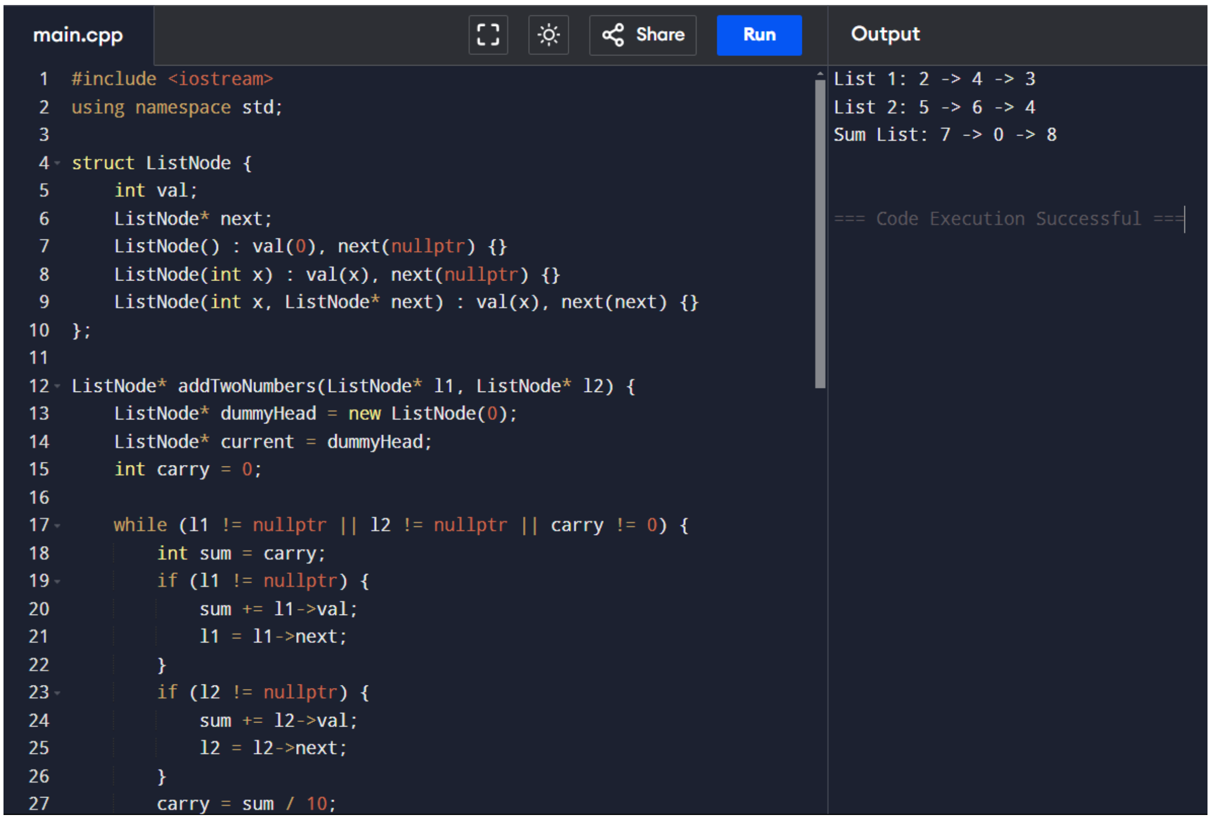
#include <iostream>

```
using namespace std;

struct ListNode {

    int val;

    ListNode* next;

    ListNode() : val(0), next(nullptr) {}

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

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

};

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* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val;
        if (head->next) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    int arr1[] = {2, 4, 3};
    int arr2[] = {5, 6, 4};

    ListNode* l1 = createList(arr1, 3);
    ListNode* l2 = createList(arr2, 3);
```

```
cout << "List 1: ";
  printList(l1);
cout << "List 2: ";
    printList(l2);
 ListNode* result = addTwoNumbers(l1, l2);
cout << "Sum List: ";
printList(result);
return 0;
}
```

Output:



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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

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;
```

```
List 1: 2 -> 4 -> 3
List 2: 5 -> 6 -> 4
Sum List: 7 -> 0 -> 8


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

Q2 . Elimination Game

You have a list arr of all integers in the range [1, n] sorted in a strictly increasing order. Apply the following algorithm on arr:

Starting from left to right, remove the first number and every other number afterward until you reach the end of the list.

Repeat the previous step again, but this time from right to left, remove the rightmost number and every other number from the remaining numbers.

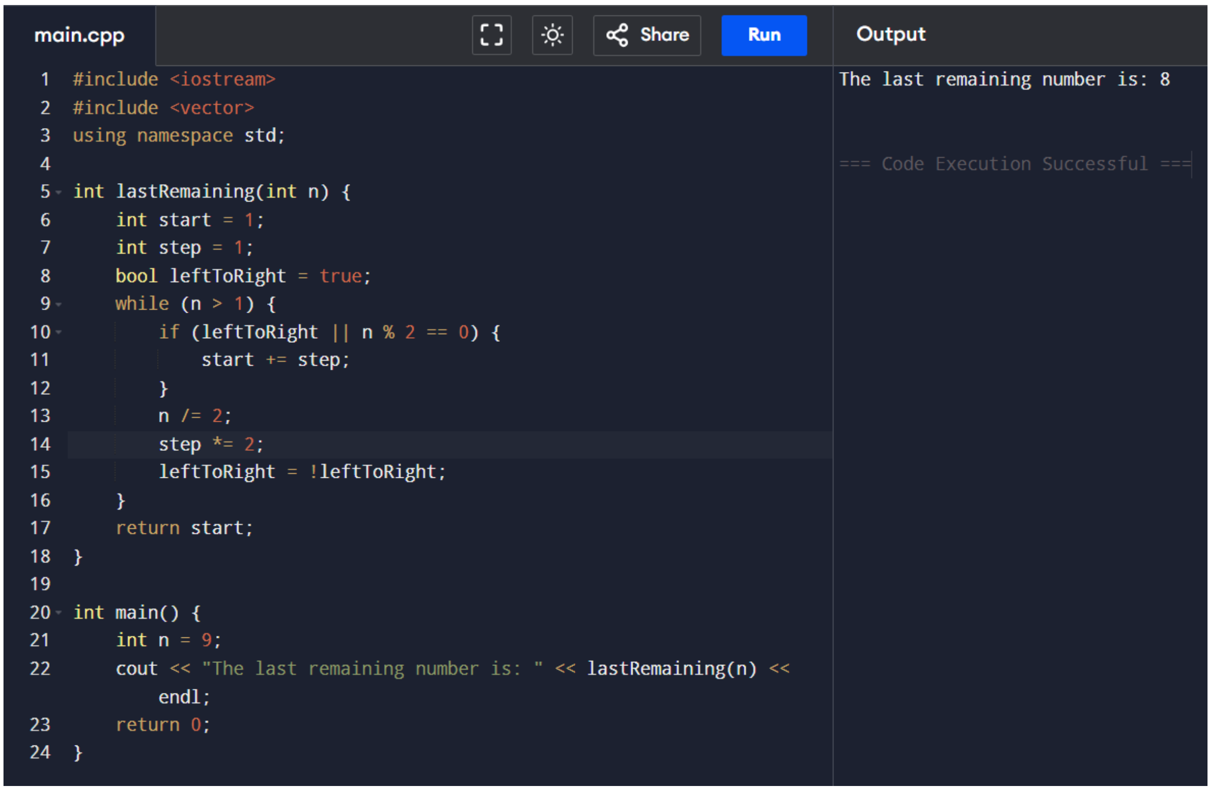
Keep repeating the steps again, alternating left to right and right to left, until a single number remains.

Given the integer n, return the last number that remains in arr.

Code:

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

int lastRemaining(int n) {
   int start = 1;
   int step = 1;
   bool leftToRight = true;
   while (n > 1) {
      if (leftToRight || n % 2 == 0) {
         start += step;
      }
      n /= 2;
      step *= 2;
      leftToRight = !leftToRight;
   }
   return start;
}

int main() {
   int n = 9;
   cout << "The last remaining number is: " << lastRemaining(n) << endl;
   return 0;
}
```

Output:



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

int lastRemaining(int n) {
    int start = 1;
    int step = 1;
    bool leftToRight = true;
    while (n > 1) {
        if (leftToRight || n % 2 == 0) {
            start += step;
        }
        n /= 2;
        step *= 2;
        leftToRight = !leftToRight;
    }
    return start;
}

int main() {
    int n = 9;
    cout << "The last remaining number is: " << lastRemaining(n) <<
        endl;
    return 0;
}
```

```
The last remaining number is: 8


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

Q3 . Predict The Winner

You are given an integer array nums. Two players are playing a game with this array: player 1 and player 2.

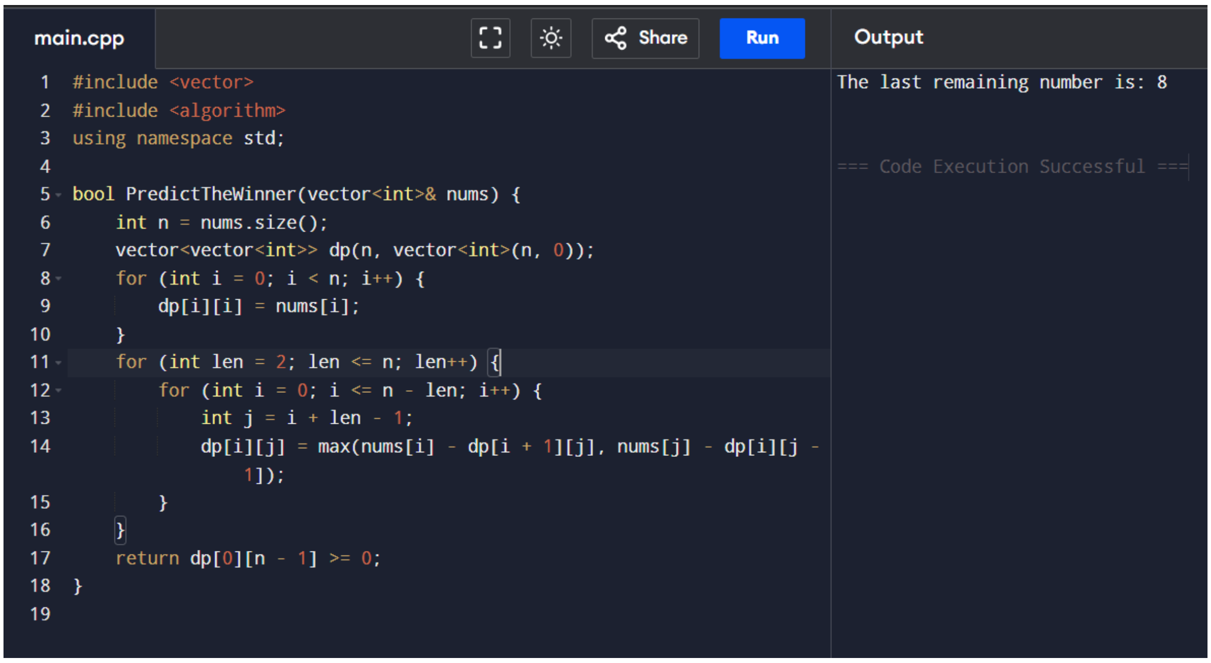
Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of 0. At each turn, the player takes one of the numbers from either end of the array (i.e., nums[0] or nums[nums.length - 1]) which reduces the size of the array by 1. The player adds the chosen number to their score. The game ends when there are no more elements in the array.

Return true if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return true. You may assume that both players are playing optimally.

Code:

```cpp
#include <vector>

#include <algorithm>

using namespace std;


bool PredictTheWinner(vector<int>& nums) {
    int n = nums.size();
    vector<vector<int>> dp(n, vector<int>(n, 0));
    for (int i = 0; i < n; i++) {
        dp[i][i] = nums[i];
    }
    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            dp[i][j] = max(nums[i]  - dp[i + 1][j],  nums[j] - dp[i][j - 1]);
        }
    }
    return dp[0][n - 1] >= 0;
}
```

Output:



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

bool PredictTheWinner(vector<int>& nums) {
    int n = nums.size();
    vector<vector<int>> dp(n, vector<int>(n, 0));
    for (int i = 0; i < n; i++) {
        dp[i][i] = nums[i];
    }
    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            dp[i][j] = max(nums[i] - dp[i + 1][j], nums[j] - dp[i][j -
                1]);
        }
    }
    return dp[0][n - 1] >= 0;
}
```

```
The last remaining number is: 8


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

Hard

Q1 ..Regular Expression Matching

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

'.' Matches any single character.

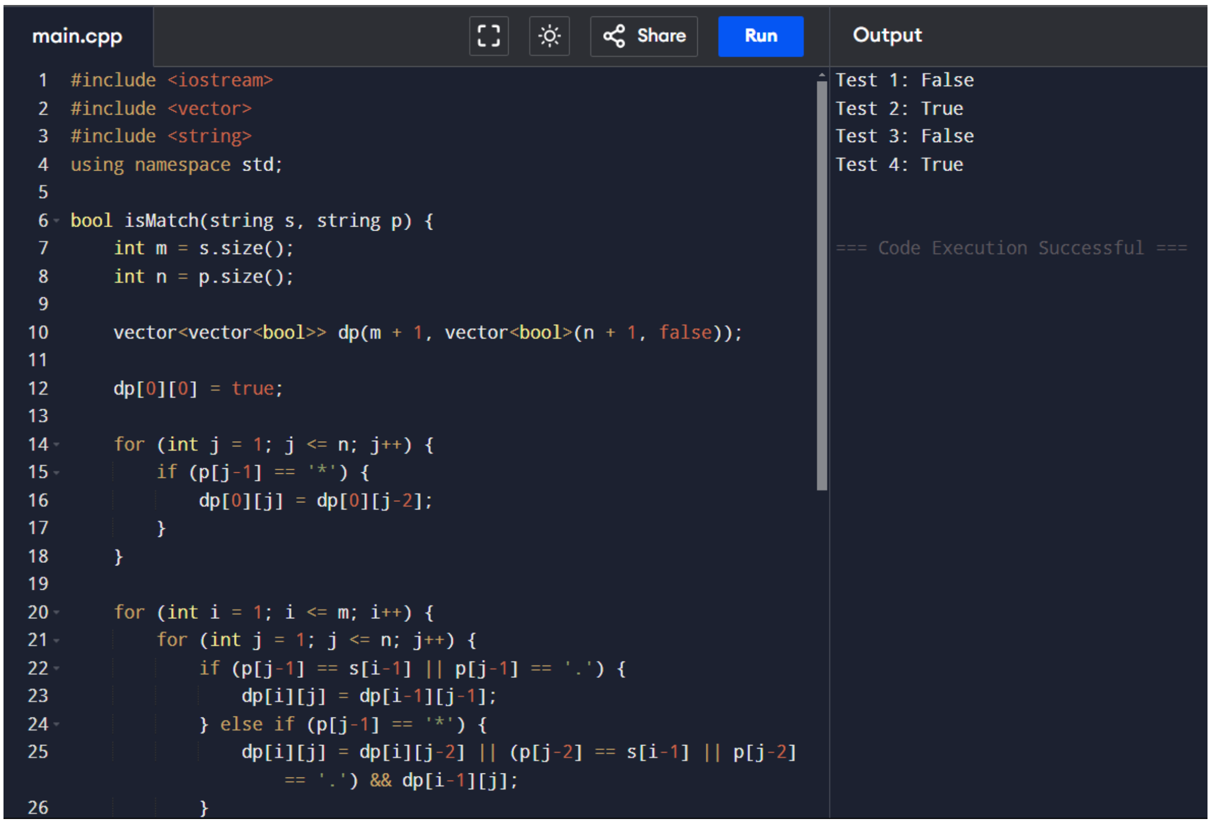
'*' Matches zero or more of the preceding element.

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

Code:

```
#include <iostream>

#include <vector>

#include <string>

using namespace std;


bool isMatch(string s, string p) {

    int m = s.size();

    int 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-2];

        }

    }
    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-2] || (p[j-2] == s[i-1] || p[j-2] == '.') && dp[i-1][j];
            }
        }
    }

    return dp[m][n];
}

int main() {
    string s1 = "aa";
    string p1 = "a";
    cout << "Test 1: " << (isMatch(s1, p1) ? "True" : "False") << endl;

    string s2 = "aa";
    string p2 = "a*";
    cout << "Test 2: " << (isMatch(s2, p2) ? "True" : "False") << endl;

    string s3 = "mississippi";
    string p3 = "mis*is*p*.";
    cout << "Test 3: " << (isMatch(s3, p3) ? "True" : "False") << endl;
    string s4 = "ab";

    string p4 = ".*";

    cout << "Test 4: " << (isMatch(s4, p4) ? "True" : "False") << endl;


    return 0;

}
```

Output:



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

bool isMatch(string s, string p) {
    int m = s.size();
    int 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-2];
        }
    }

    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-2] || (p[j-2] == s[i-1] || p[j-2]
                    == '.') && dp[i-1][j];
            }
```

```
Test 1: False
Test 2: True
Test 3: False
Test 4: True


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

Q2. Reverse Nodes in k-Group

Given the head of a linked list, reverse the nodes of the list k at a time, and return the modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:

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

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

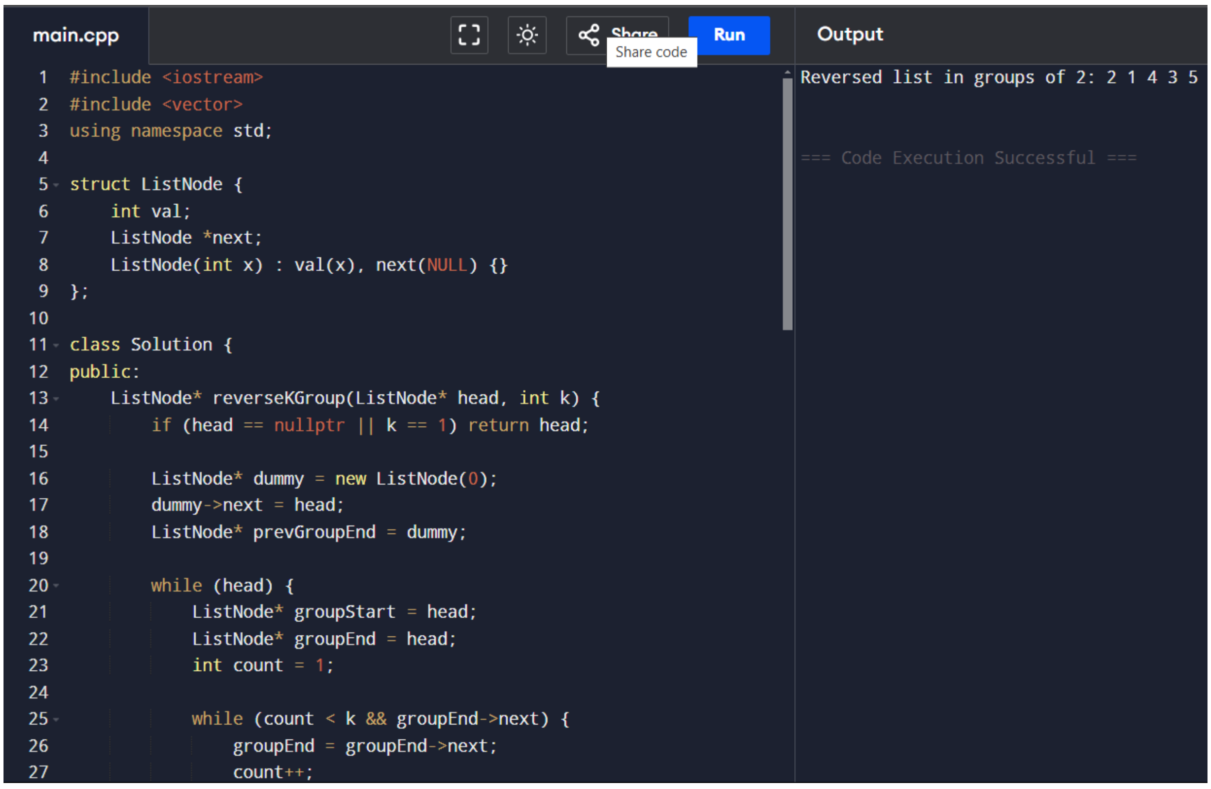
Code:

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

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

class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        if (head == nullptr || k == 1) return head;

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prevGroupEnd = dummy;

        while (head) {
            ListNode* groupStart = head;
            ListNode* groupEnd = head;
            int count = 1;

            while (count < k && groupEnd->next) {
                groupEnd = groupEnd->next;
                count++;
            }

            if (count < k) break;

            ListNode* nextGroupStart = groupEnd->next;
            ListNode* prev = nextGroupStart;
            ListNode* curr = groupStart;
            while (curr != nextGroupStart) {
                ListNode* temp = curr->next;
                curr->next = prev;
                prev = curr;
                curr = temp;
            }

            prevGroupEnd->next = groupEnd;
            groupStart->next = nextGroupStart;

            prevGroupEnd = groupStart;
            head = nextGroupStart;
        }
```

```
        return dummy->next;
    }
};

ListNode* createList(const vector<int>& nums) {
    ListNode* head = nullptr;
    ListNode* tail = nullptr;
    for (int num : nums) {
        ListNode* newNode = new ListNode(num);
        if (tail) {
            tail->next = newNode;
        } else {
            head = newNode;
        }
        tail = newNode;
    }
    return head;
}

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

```
int main() {
    Solution solution;

    vector<int> nums = {1, 2, 3, 4, 5};
    int k = 2;
    ListNode* head = createList(nums);
    ListNode* result = solution.reverseKGroup(head, k);
    cout << "Reversed list in groups of " << k << ": ";
    printList(result);

    return 0;
}
```

Output:



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

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

class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        if (head == nullptr || k == 1) return head;

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prevGroupEnd = dummy;

        while (head) {
            ListNode* groupStart = head;
            ListNode* groupEnd = head;
            int count = 1;

            while (count < k && groupEnd->next) {
                groupEnd = groupEnd->next;
                count++;
```

```
Reversed list in groups of 2: 2 1 4 3 5


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

Q3. Wildcard Matching

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

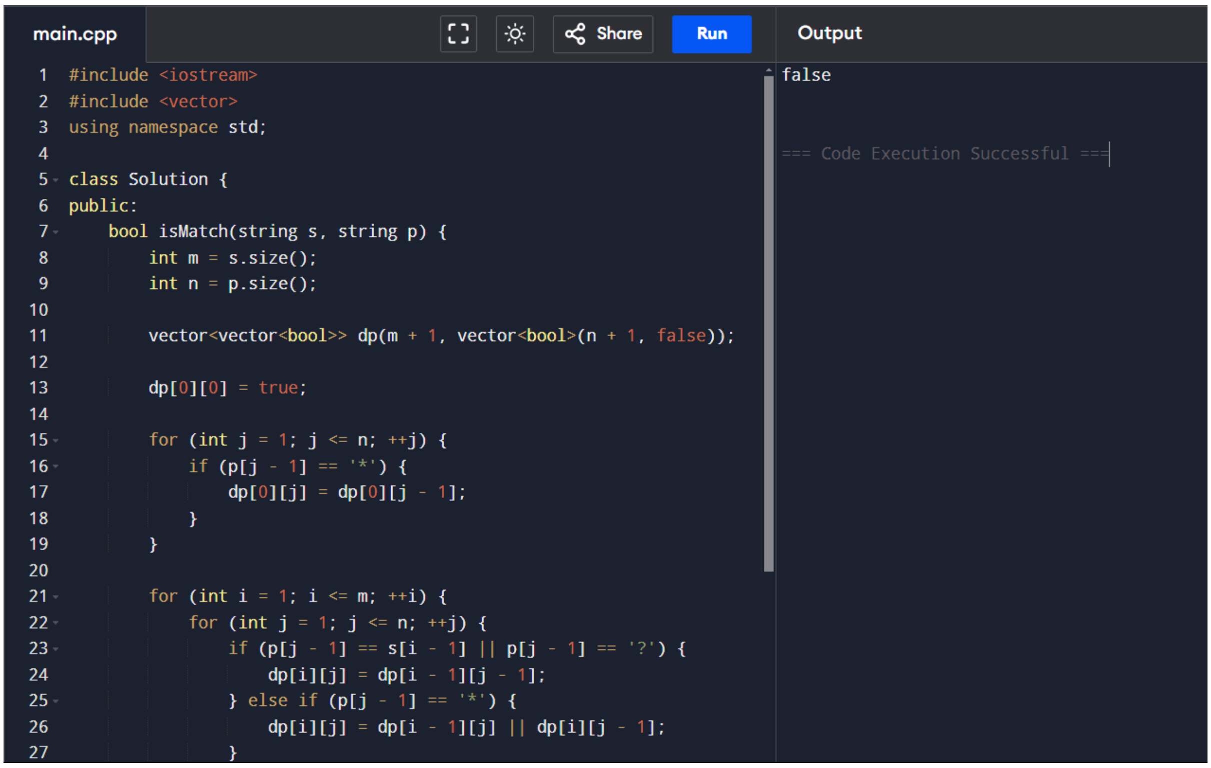
'?' Matches any single character.

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

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

```
class Solution {
public:
    bool isMatch(string s, string p) {
        int m = s.size();
        int 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() {
    Solution solution;
    string s = "aa";
    string p = "a";
    bool result = solution.isMatch(s, p);
    cout << (result ? "true" : "false") << endl;

    return 0;
}
```

Output:



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

class Solution {
public:
    bool isMatch(string s, string p) {
        int m = s.size();
        int 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];
                }
```

```
false


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

Very Hard

Q1. Find the K-th Character in String Game II

Alice and Bob are playing a game. Initially, Alice has a string word = "a".

You are given a positive integer k. You are also given an integer array operations, where operations[i] represents the type of the ith operation.

Now Bob will ask Alice to perform all operations in sequence:

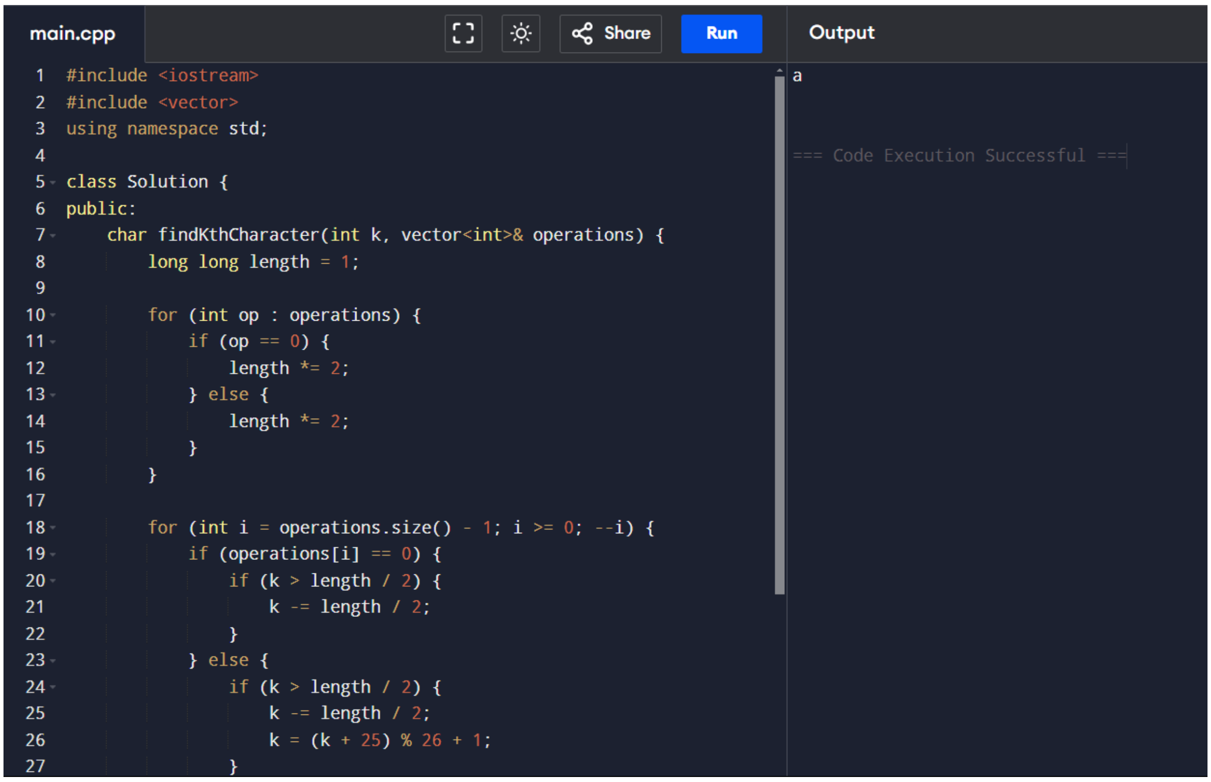
If operations[i] == 0, append a copy of word to itself.

.

Code:

```
#include <iostream>

#include <vector>

using namespace std;


class Solution {

public:
    char findKthCharacter(int k, vector<int>& operations) {
        long long length = 1;

        for (int op : operations) {
            if (op == 0) {
                length *= 2;
            } else {
                length *= 2;
            }
        }

        for (int i = operations.size() - 1; i >= 0; --i) {
            if (operations[i] == 0) {
                if (k > length / 2) {
                    k -= length / 2;
                }
            } else {
                if (k > length / 2) {
                    k -= length / 2;
                    k = (k + 25) % 26 + 1;
                }
            }
            length /= 2;
        }

        return 'a' + (k - 1);
    }
};

int main() {
    Solution solution;
    vector<int> operations = {0, 0, 0};
    int k = 5;
    cout << solution.findKthCharacter(k, operations) << endl;
    return 0;
}
```

Output:



main.cpp

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

class Solution {
public:
    char findKthCharacter(int k, vector<int>& operations) {
        long long length = 1;

        for (int op : operations) {
            if (op == 0) {
                length *= 2;
            } else {
                length *= 2;
            }
        }

        for (int i = operations.size() - 1; i >= 0; --i) {
            if (operations[i] == 0) {
                if (k > length / 2) {
                    k -= length / 2;
                }
            } else {
                if (k > length / 2) {
                    k -= length / 2;
                    k = (k + 25) % 26 + 1;
                }
```

Output

```
a


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

Q2. Maximize Number of Nice Divisors

You are given a positive integer primeFactors. You are asked to construct a positive integer n that satisfies the following conditions:

The number of prime factors of n (not necessarily distinct) is at most primeFactors.

The number of nice divisors of n is maximized. Note that a divisor of n is nice if it is divisible by every prime factor of n. For example, if n = 12, then its prime factors are [2,2,3], then 6 and 12 are nice divisors, while 3 and 4 are not.

Return the number of nice divisors of n. Since that number can be too large, return it modulo 109 + 7.

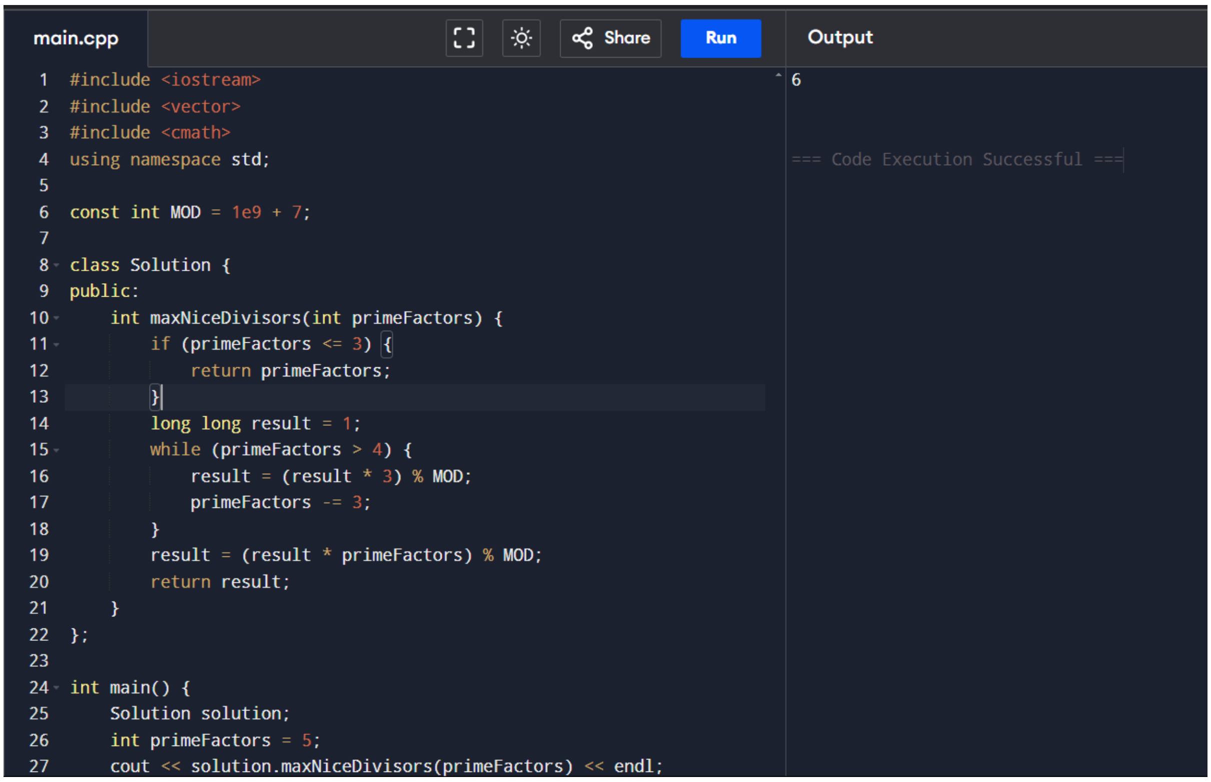
Note that a prime number is a natural number greater than 1 that is not a product of two smaller natural numbers. The prime factors of a number n is a list of prime numbers such that their product equals n.

Code:

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

const int MOD = 1e9 + 7;

class Solution {
public:
   int maxNiceDivisors(int primeFactors) {
      if (primeFactors <= 3) {
         return primeFactors;
      }
      long long result = 1;
      while (primeFactors > 4) {
         result = (result * 3) % MOD;
         primeFactors -= 3;
      }
      result = (result * primeFactors) % MOD;
      return result;
   }
};

int main() {
   Solution solution;
```

```
    int primeFactors = 5;

    cout << solution.maxNiceDivisors(primeFactors)  << endl;

    return 0;

}
```

Output:



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

const int MOD = 1e9 + 7;

class Solution {
public:
    int maxNiceDivisors(int primeFactors) {
        if (primeFactors <= 3) {
            return primeFactors;
        }
        long long result = 1;
        while (primeFactors > 4) {
            result = (result * 3) % MOD;
            primeFactors -= 3;
        }
        result = (result * primeFactors) % MOD;
        return result;
    }
};

int main() {
    Solution solution;
    int primeFactors = 5;
    cout << solution.maxNiceDivisors(primeFactors) << endl;
```

```
6


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

Q3. Parsing a Boolean Expression

A boolean expression is an expression that evaluates to either true or false. It can be in one of the following shapes:

't' that evaluates to true.

'f' that evaluates to false.

'!(subExpr)' that evaluates to the logical NOT of the inner expression subExpr.

'&(subExpr1, subExpr2, ..., subExprn)' that evaluates to the logical AND of the inner expressions subExpr1, subExpr2, ..., subExprn where n >= 1.

'|(subExpr1, subExpr2, ..., subExprn)' that evaluates to the logical OR of the inner expressions subExpr1, subExpr2, ..., subExprn where n >= 1.

Given a string expression that represents a boolean expression, return the evaluation of that expression.

It is guaranteed that the given expression is valid and follows the given rules.

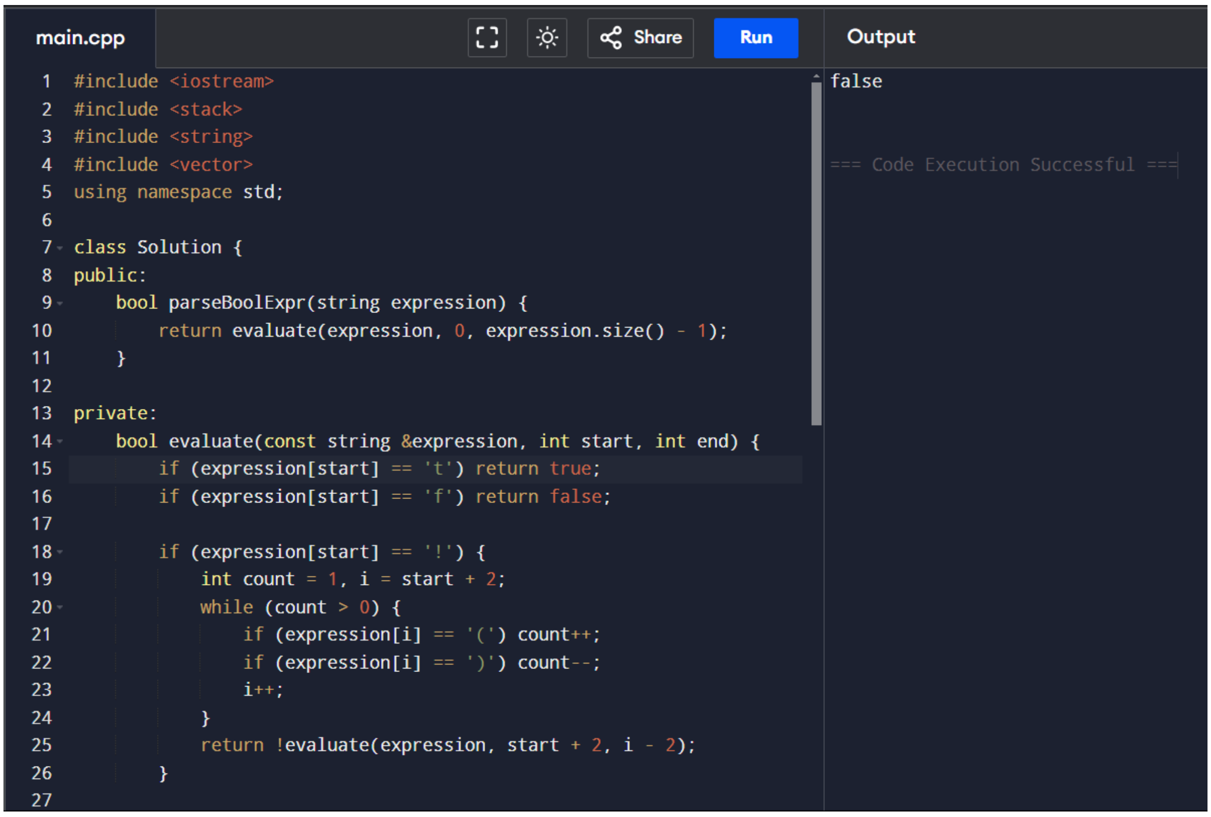
Code:

```
#include <iostream>

#include <stack>

#include <string>

#include <vector>

using namespace std;


class Solution {

public:

    bool parseBoolExpr(string expression) {

        return evaluate(expression, 0, expression.size() - 1);

    }


private:

    bool evaluate(const string &expression, int start, int end) {

        if (expression[start] == 't') return true;

        if (expression[start] == 'f') return false;


        if (expression[start] == '!') {

            int count = 1, i = start + 2;

            while (count > 0) {
```

```
        if (expression[i] == '(') count++;

        if (expression[i] == ')') count--;

        i++;

    }

    return !evaluate(expression, start + 2, i - 2);

}


char operation = expression[start];

vector<bool> subExprResults;


int count = 1, i = start + 2;

int subExprStart = i;

while (count > 0) {

    if (expression[i] == '(') count++;

    if (expression[i] == ')') count--;

    if (expression[i] == ',' && count == 1) {

        subExprResults.push_back(evaluate(expression, subExprStart, i - 1));

        subExprStart = i + 1;

    }

    i++;

}

subExprResults.push_back(evaluate(expression, subExprStart, i - 2));


if (operation == '&') {

    for (bool result : subExprResults) {

        if (!result) return false;
```

```
            }
            return true;
        } else {
            for (bool result : subExprResults) {
                if (result) return true;
            }
            return false;
        }
    }
};

int main() {
    Solution solution;
    string expression = "&(|(f))";
    cout << boolalpha << solution.parseBoolExpr(expression)  << endl;
    return 0;
}
```

Output:



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

class Solution {
public:
    bool parseBoolExpr(string expression) {
        return evaluate(expression, 0, expression.size() - 1);
    }

private:
    bool evaluate(const string &expression, int start, int end) {
        if (expression[start] == 't') return true;
        if (expression[start] == 'f') return false;

        if (expression[start] == '!') {
            int count = 1, i = start + 2;
            while (count > 0) {
                if (expression[i] == '(') count++;
                if (expression[i] == ')') count--;
                i++;
            }
            return !evaluate(expression, start + 2, i - 2);
        }

```

```
false


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