TCS APRIL-MAY 2024 QUESTION LIST

Date- 26 April Shift-1

Q1 -Find Sub-Arrays with a given sum in array. Given integer array find subarray with a given sum in it.

I/P- arr = [3,4,-7,1,3,3,1,-4] target= 7

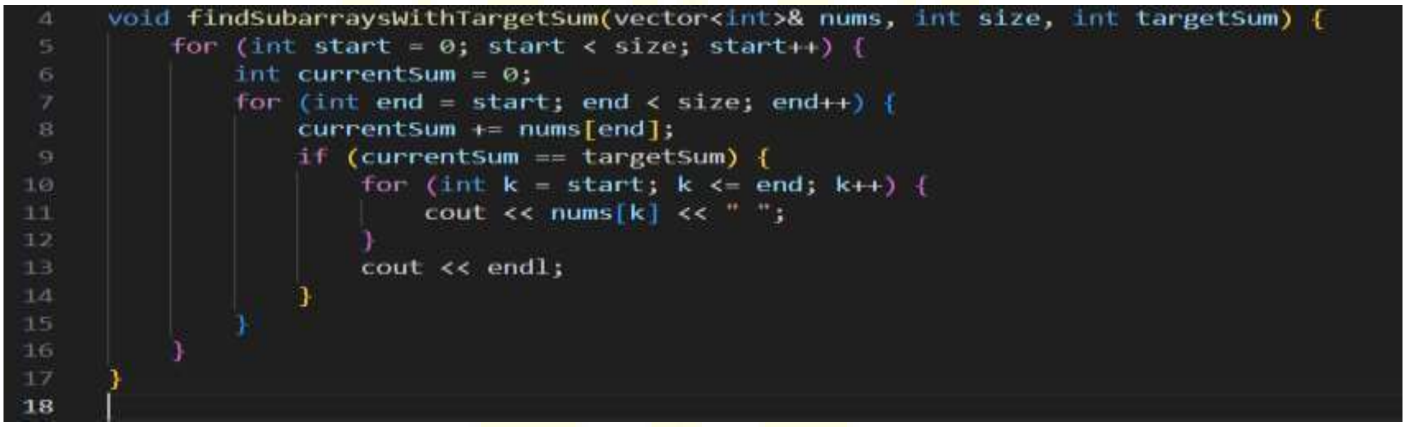
O/P- [3,4] , [3,4,-7,1,3,3] , [1,3,3] , [3,3,1]

Approach-

Brute Force Approach is simple , we directly use two for loops to find sub Array and third loop to print the required Array.

In Optimized approach uses a hash map to efficiently find subarrays that sum to a target value. It maintains a cumulative sum (curSum) as it iterates through the array. For each element, it checks if curSum equals the target or if curSum - target is present in the hash map. If so, it prints the subarray that sums to the target. The hash map stores the cumulative sum and its corresponding index, allowing for quick lookups and subarray identification.

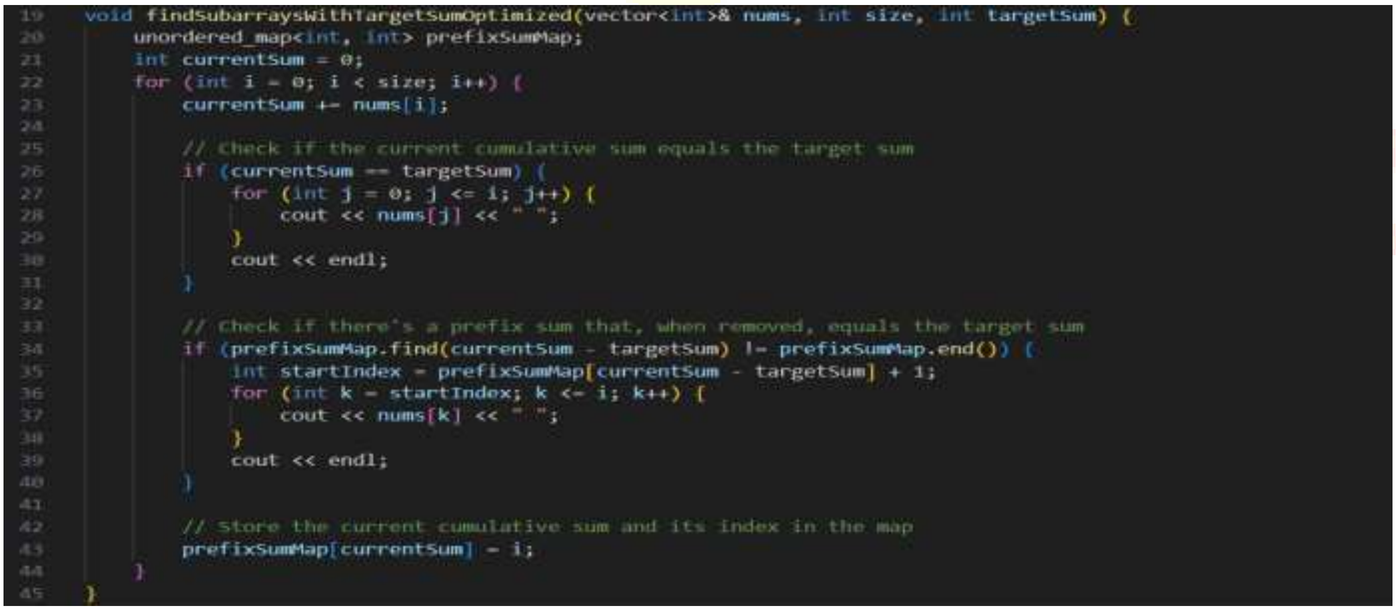
Brute Force-



```
void findSubarraysWithTargetSum(vector<int>& nums, int size, int targetSum) {
    for (int start = 0; start < size; start++) {
        int currentSum = 0;
        for (int end = start; end < size; end++) {
            currentSum += nums[end];
            if (currentSum == targetSum) {
                for (int k = start; k <= end; k++) {
                    cout << nums[k] << " ";
                }
                cout << endl;
            }
        }
    }
}

```

Optimized Solution-



```
void findSubarraysWithTargetSumOptimized(vector<int>& nums, int size, int targetSum) {
    unordered_map<int, int> prefixSumMap;
    int currentSum = 0;
    for (int i = 0; i < size; i++) {
        currentSum += nums[i];

        // Check if the current cumulative sum equals the target sum
        if (currentSum == targetSum) {
            for (int j = 0; j <= i; j++) {
                cout << nums[j] << " ";
            }
            cout << endl;
        }

        // Check if there's a prefix sum that, when removed, equals the target sum
        if (prefixSumMap.find(currentSum - targetSum) != prefixSumMap.end()) {
            int startIndex = prefixSumMap[currentSum - targetSum] + 1;
            for (int k = startIndex; k <= i; k++) {
                cout << nums[k] << " ";
            }
            cout << endl;
        }

        // Store the current cumulative sum and its index in the map
        prefixSumMap[currentSum] = i;
    }
}
```

Q2 - There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]).

The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]).

The robot can only move either down or right at any point in time.

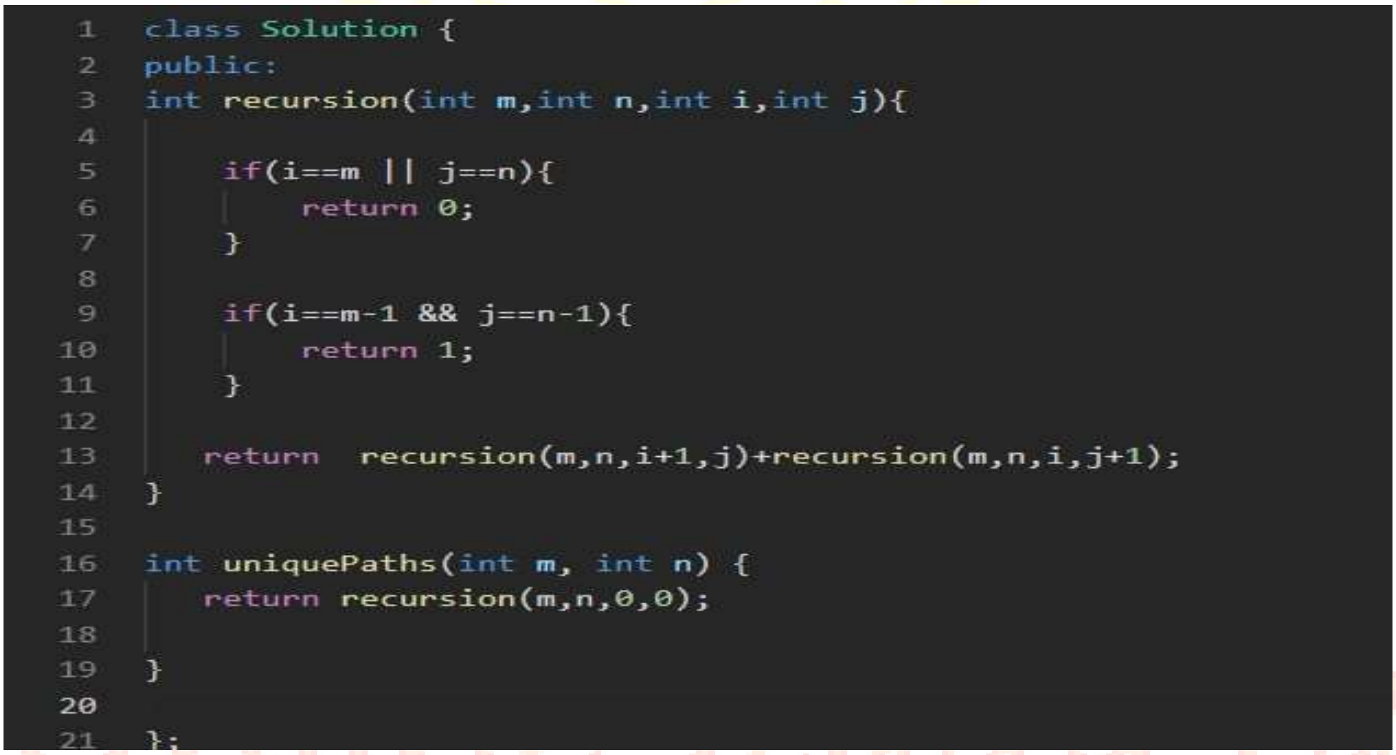
Given the two integers m and n, return the number of possible unique paths that the robot can take to reach

the bottom-right corner.

I/O: m = 3, n = 2

O/P: 3

Approach-
This approach calculates the number of unique paths in an m x n grid using recursion. Starting from the top-left corner (0,0), it recursively explores moving right and down until it reaches the bottom-right corner (m-1, n-1). The base cases handle reaching the grid boundaries or the destination. The function sums the paths from moving right and down to compute the total number of unique paths.



```
class Solution {
public:
int recursion(int m,int n,int i,int j){

    if(i==m || j==n){
        return 0;
    }

    if(i==m-1 && j==n-1){
        return 1;
    }

   return  recursion(m,n,i+1,j)+recursion(m,n,i,j+1);
}

int uniquePaths(int m, int n) {
   return recursion(m,n,0,0);

}

};
```

Date- 26 April S2

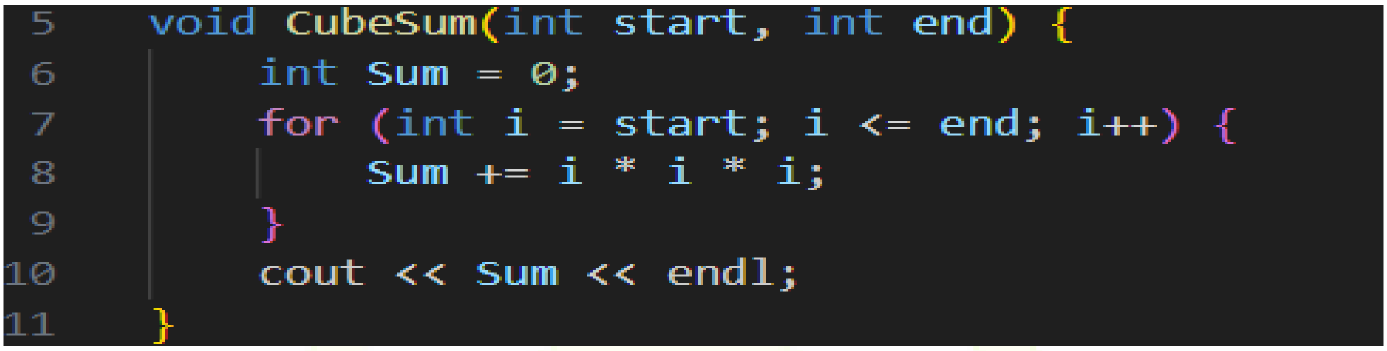
Q1 - Given two Integer, find sum of cubes of all numbers in the range of n & m.

I/P- n=1 , m=13

O/P- 8281

Approach-

The CubeSum function calculates the sum of cubes of all integers from start to end (inclusive). It iterates through each integer in this range, computes its cube, and accumulates these values in the variable Sum. After completing the iteration, it prints the total sum.



```cpp
void CubeSum(int start, int end) {
    int Sum = 0;
    for (int i = start; i <= end; i++) {
        Sum += i * i * i;
    }
    cout << Sum << endl;
}
```

Q2-You are given a grocery List which consists of three parameters ITEMS, QUANTITY,PRICE.

Your task is to find 1)Higher Selling item 2)Total Selling item 3)Avg. Selling item

I/P- Pen ,20,10

Notebook,100,10

Pencil,10,10

O/P- Notebook , 1300,433.33

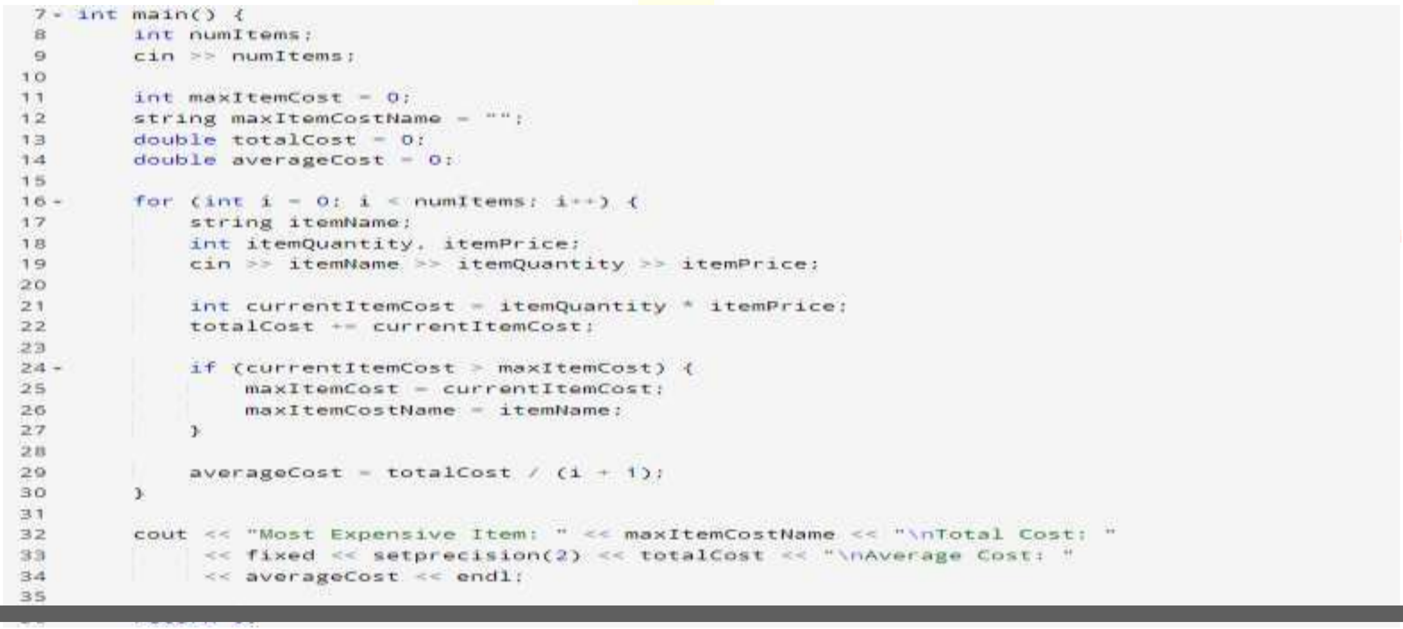
Approach-

1. Input and Calculation: The program reads item details, calculates each item's total cost, and updates the overall total cost.

2. Maximum Cost Tracking: It keeps track of the item with the highest total cost.

3. Average Cost: It computes the average cost by dividing the total cost by the number of items.

4. Output: Finally, it prints the most expensive item, the total cost formatted to two decimal places, and the average cost.



```cpp
int main() {
    int numItems;
    cin >> numItems;

    int maxItemCost = 0;
    string maxItemCostName = "";
    double totalCost = 0;
    double averageCost = 0;

    for (int i = 0; i < numItems; i++) {
        string itemName;
        int itemQuantity, itemPrice;
        cin >> itemName >> itemQuantity >> itemPrice;

        int currentItemCost = itemQuantity * itemPrice;
        totalCost += currentItemCost;

        if (currentItemCost > maxItemCost) {
            maxItemCost = currentItemCost;
            maxItemCostName = itemName;
        }

        averageCost = totalCost / (i + 1);
    }

    cout << "Most Expensive Item: " << maxItemCostName << "\nTotal Cost: "
         << fixed << setprecision(2) << totalCost << "\nAverage Cost: "
         << averageCost << endl;

```

Date- 29 April S1

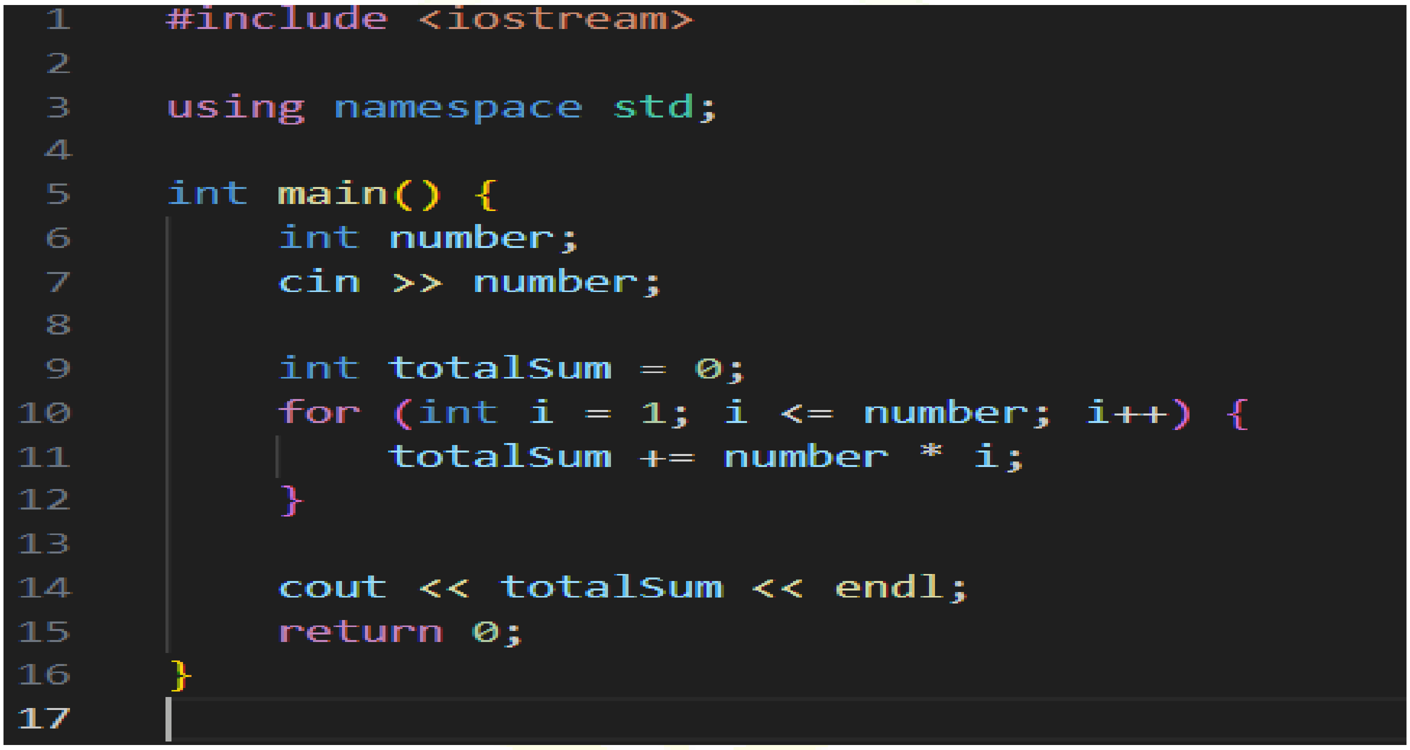
Q1- Given an Integer , we need to find the sum of values of that table.

I/P- 12

O/P- 660

Approach-

The program reads an integer `number` from the user and then calculates the total sum of `number` multiplied by each integer from 1 up to `number`. It does this by iterating through values from 1 to `number`, adding the product of `number` and each iterator value `i` to a running total. After completing the iteration, it prints the total sum, which represents the sum of all these products.



```cpp
#include <iostream>

using namespace std;

int main() {
    int number;
    cin >> number;

    int totalSum = 0;
    for (int i = 1; i <= number; i++) {
        totalSum += number * i;
    }

    cout << totalSum << endl;
    return 0;
}

```

Q2-Given an Array and a integer k , we need to find the maximum element in each of the contiguous subarrays.

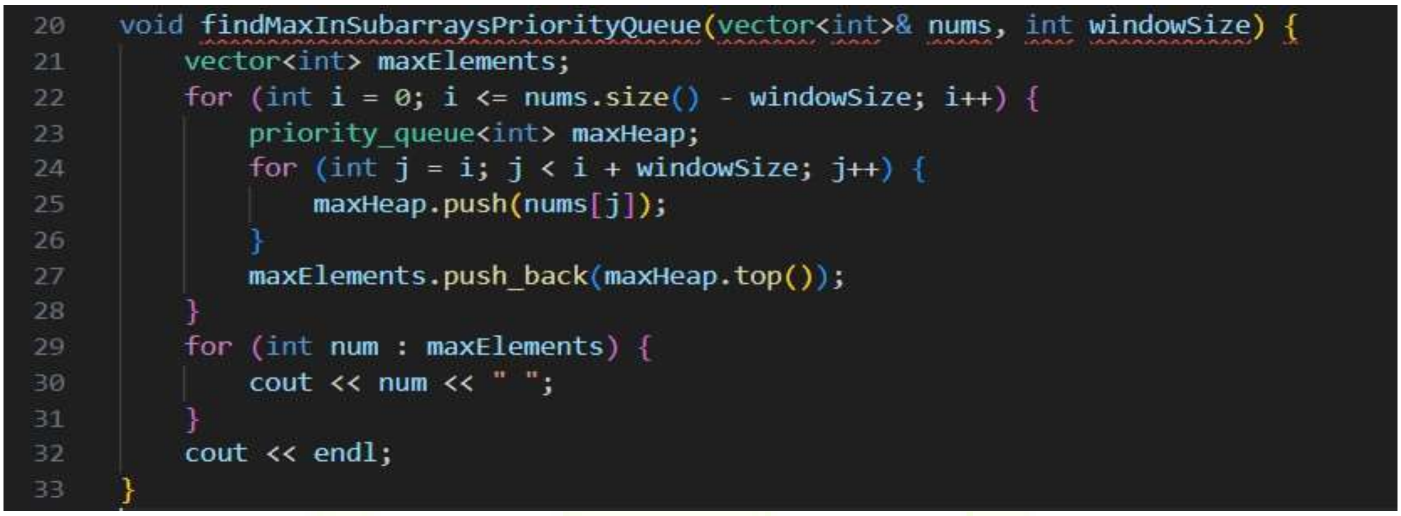
I/P- [2,3,6,1,5,1]

k=3

O/P- [6,6,6,5]

Approach-

The function finds the maximum element in each subarray of size windowSize from a given array. It does this by iterating through the array and for each subarray, it uses a max heap (priority queue) to determine the maximum value within that subarray. The maximum value is then stored in a result vector. Finally, the function prints all the maximum values for each subarray.



```cpp
void findMaxInSubarraysPriorityQueue(vector<int>& nums, int windowSize) {
    vector<int> maxElements;
    for (int i = 0; i <= nums.size() - windowSize; i++) {
        priority_queue<int> maxHeap;
        for (int j = i; j < i + windowSize; j++) {
            maxHeap.push(nums[j]);
        }
        maxElements.push_back(maxHeap.top());
    }
    for (int num : maxElements) {
        cout << num << " ";
    }
    cout << endl;
}
```

Date- 29 April S2

Q1-Calculate the sum of N terms if Fibonacci Series

Note: fib(0)=1

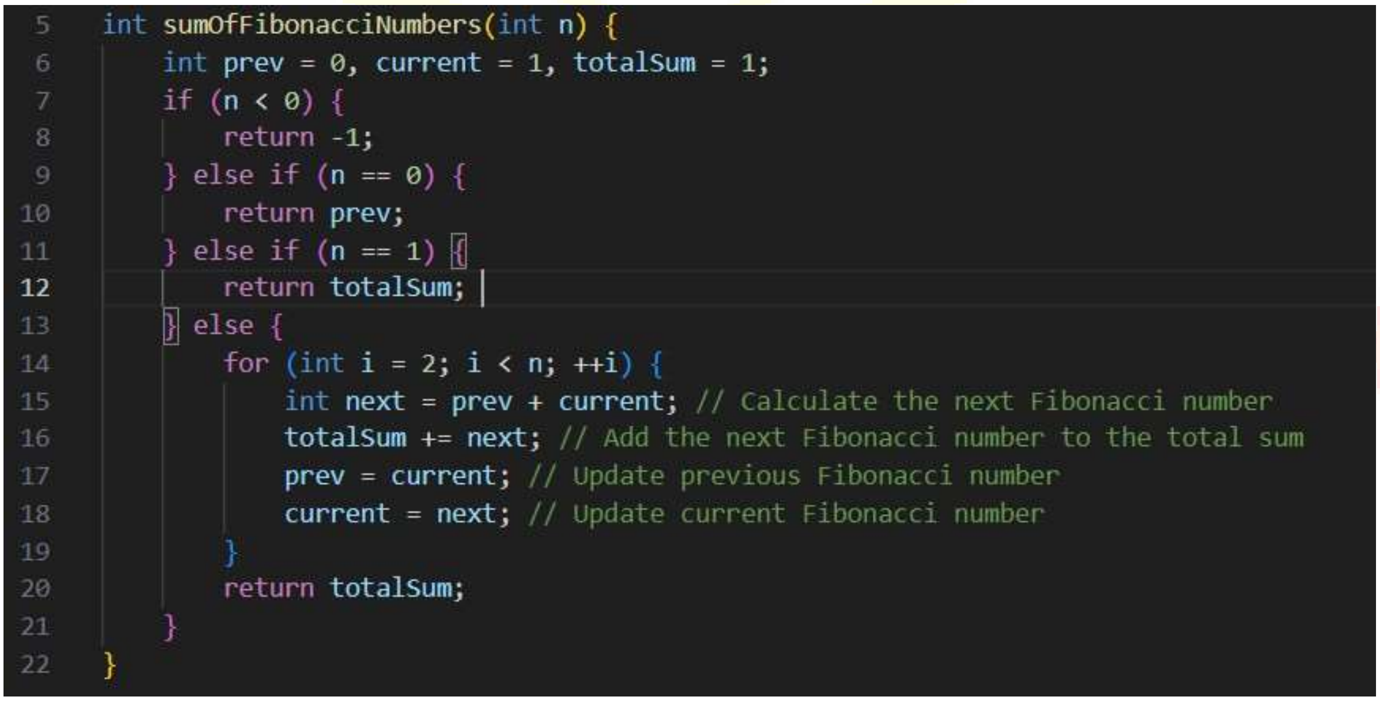
fib(1)=1

I/P: n=7

O/P: 33

Approach-

The function calculates the sum of Fibonacci numbers up to the n-th position. It initializes the first two Fibonacci numbers and starts the sum with 1. In a loop, it computes each subsequent Fibonacci number, adds it to the total sum, and updates the previous and current Fibonacci numbers accordingly. The final result is the total sum of Fibonacci numbers up to the n-th position. If n is less than 0, it returns -1 as an error code.



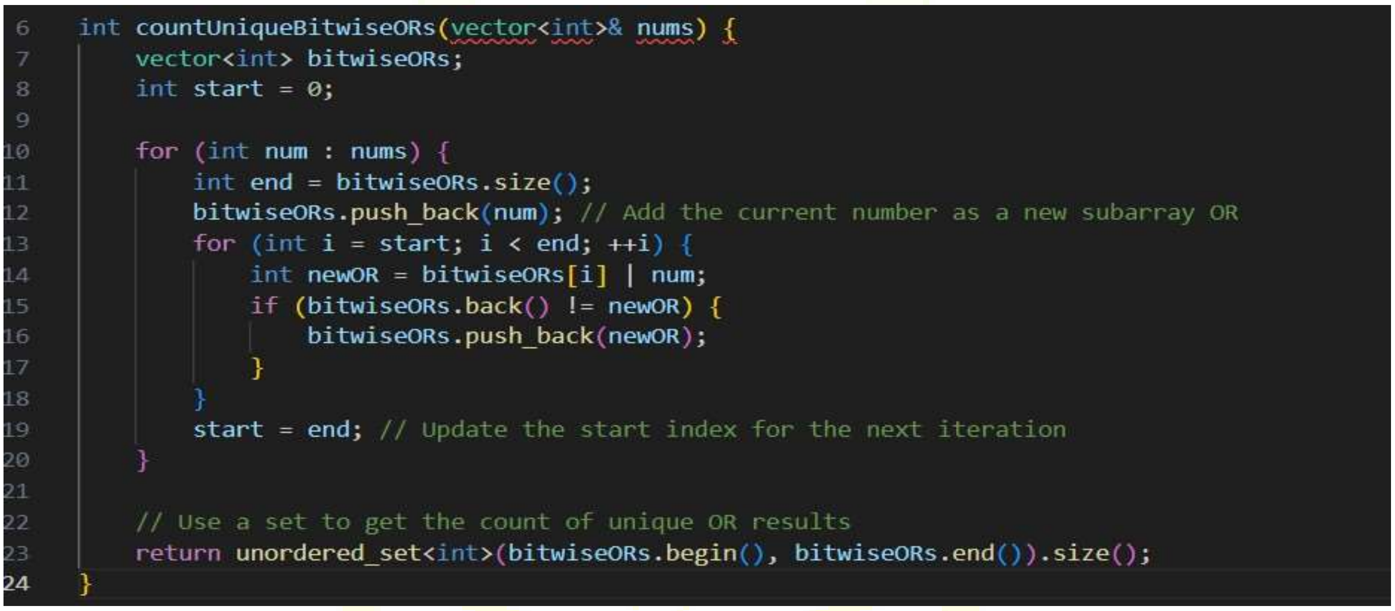
```cpp
int sumOfFibonacciNumbers(int n) {
    int prev = 0, current = 1, totalSum = 1;
    if (n < 0) {
        return -1;
    } else if (n == 0) {
        return prev;
    } else if (n == 1) {
        return totalSum;
    } else {
        for (int i = 2; i < n; ++i) {
            int next = prev + current; // Calculate the next Fibonacci number
            totalSum += next; // Add the next Fibonacci number to the total sum
            prev = current; // Update previous Fibonacci number
            current = next; // Update current Fibonacci number
        }
        return totalSum;
    }
}
```

Q2- Given an integer array arr, return the number of distinct bitwise ORs of all the non-empty subarrays of arr.The bitwise OR of a subarray is the bitwise OR of each integer in the subarray. The bitwise OR of a subarray of one integer is that integer.

I/P: arr = [1,1,2]

O/P: 3

Approach-
The function calculates the number of unique results of bitwise OR operations on all possible subarrays of a given array. It iterates through each element, updating a list of bitwise OR results by combining the current number with each existing result. This ensures that all possible subarray OR values are considered. Finally, it uses a set to determine and return the number of unique bitwise OR results.



```
int countUniqueBitwiseORs(vector<int>& nums) {
    vector<int> bitwiseORs;
    int start = 0;

    for (int num : nums) {
        int end = bitwiseORs.size();
        bitwiseORs.push_back(num); // Add the current number as a new subarray OR
        for (int i = start; i < end; ++i) {
            int newOR = bitwiseORs[i] | num;
            if (bitwiseORs.back() != newOR) {
                bitwiseORs.push_back(newOR);
            }
        }
        start = end; // Update the start index for the next iteration
    }

    // Use a set to get the count of unique OR results
    return unordered_set<int>(bitwiseORs.begin(), bitwiseORs.end()).size();
}
```

Date- 30 April S1

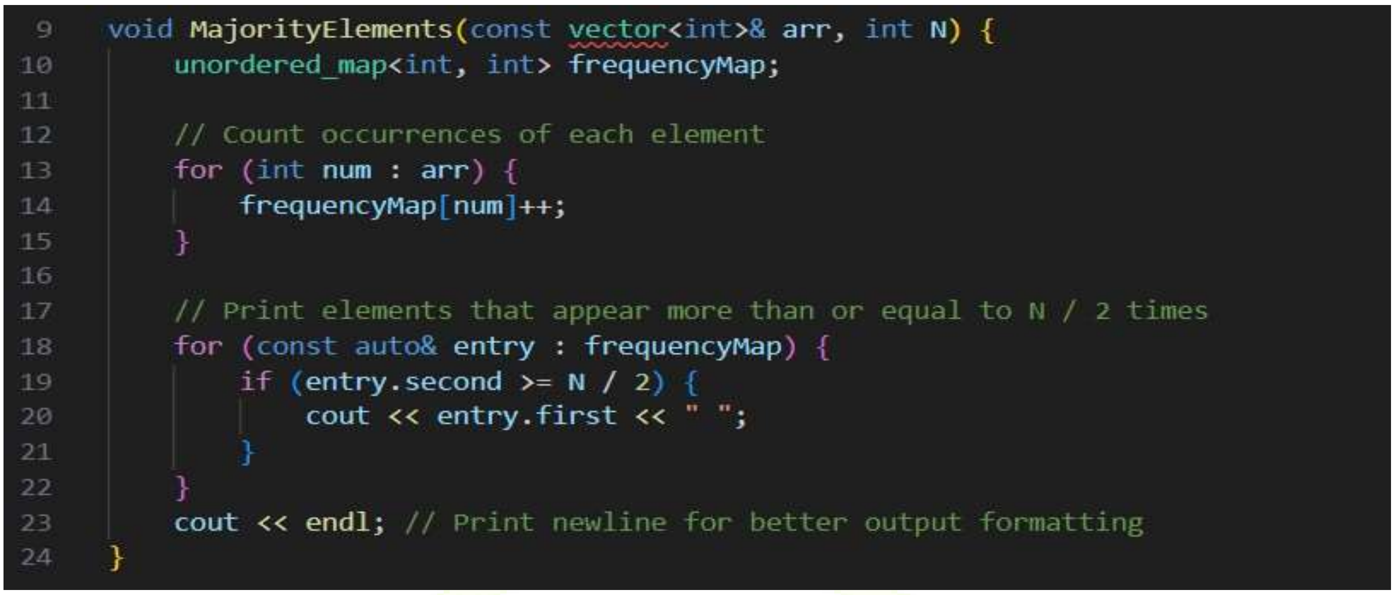
Q1- Find majority elements, where majority elements is the element which occurs more than or equal to N/2 times the array, where 'N' is the size of the array.

I/P- 8

[5,5,4,4,5,4,5,4]

O/P- 5,4

Approach-
The function identifies and prints elements in an array that appear at least \(N/2 \) times, where N/2 is the size of the array. It uses an unordered map to count the frequency of each element. After populating the map, it iterates through the map entries and prints elements that meet or exceed the frequency threshold of N/2 . This approach ensures that only majority elements are output.


```cpp
void MajorityElements(const vector<int>& arr, int N) {
    unordered_map<int, int> frequencyMap;

    // Count occurrences of each element
    for (int num : arr) {
        frequencyMap[num]++;
    }

    // Print elements that appear more than or equal to N / 2 times
    for (const auto& entry : frequencyMap) {
        if (entry.second >= N / 2) {
            cout << entry.first << " ";
        }
    }
    cout << endl; // Print newline for better output formatting
}
```

Q2-In a database there are N students, the fields of the table are NAME,AGE,GENDER. Your task is to return

the students Name who are greater than 20 years old & Calculate the average of grades using ascii value of female Candidates

I/P - 3

AA 21 A Female

BB 22 B Male

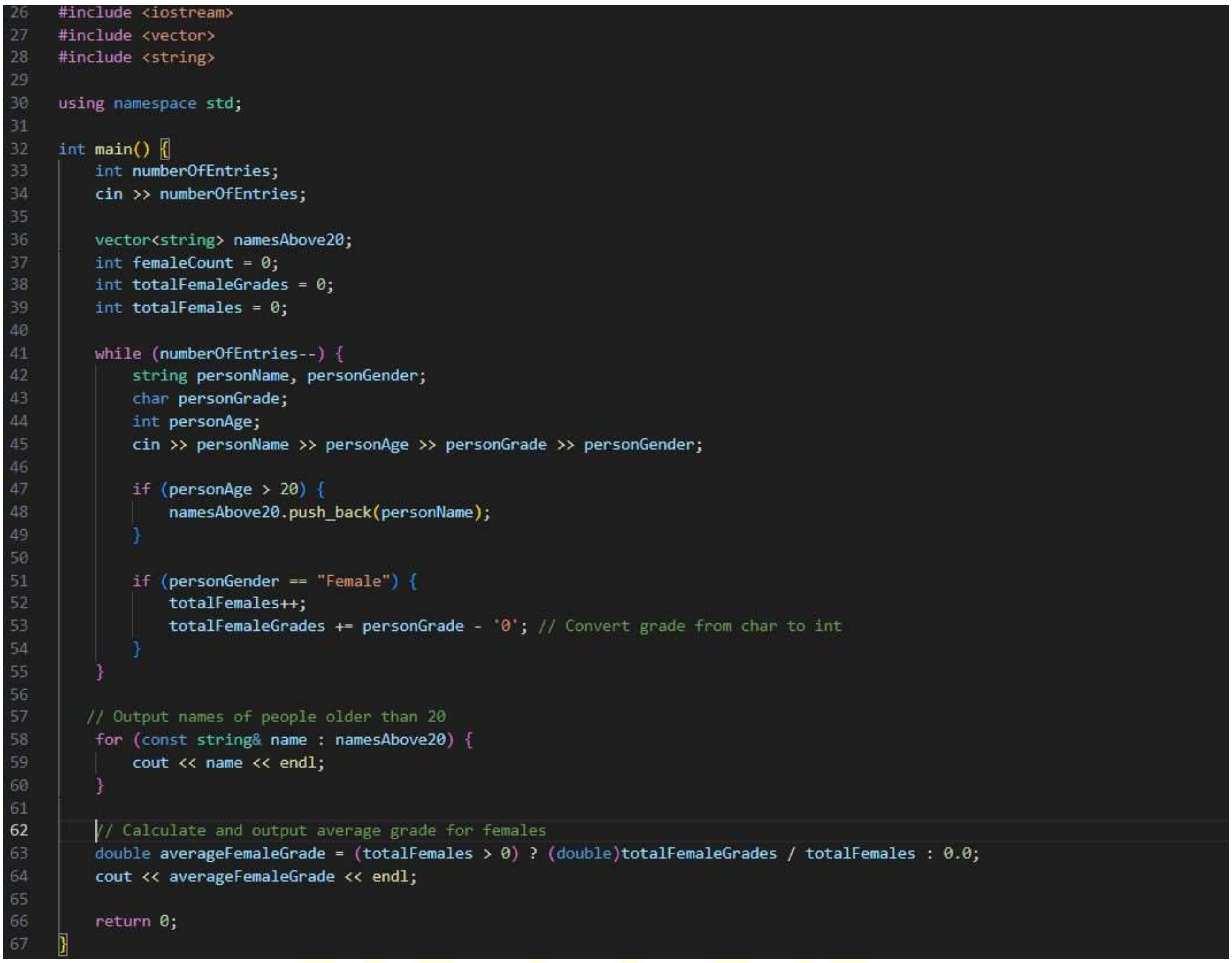
CC 24 C Female

O/P - AAA BBB CCC

66

Approach-

The program processes a list of people and performs two tasks. First, it collects and prints names of individuals older than 20. Second, it calculates the average grade for female individuals by converting their grades from characters to integers, summing them up, and then dividing by the number of females. The results are printed at the end, showing names of older individuals and the average female grade.



```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main() {
    int numberOfEntries;
    cin >> numberOfEntries;

    vector<string> namesAbove20;
    int femaleCount = 0;
    int totalFemaleGrades = 0;
    int totalFemales = 0;

    while (numberOfEntries--) {
        string personName, personGender;
        char personGrade;
        int personAge;
        cin >> personName >> personAge >> personGrade >> personGender;

        if (personAge > 20) {
            namesAbove20.push_back(personName);
        }

        if (personGender == "Female") {
            totalFemales++;
            totalFemaleGrades += personGrade - '0'; // Convert grade from char to int
        }
    }

   // Output names of people older than 20
    for (const string& name : namesAbove20) {
        cout << name << endl;
    }

    // Calculate and output average grade for females
    double averageFemaleGrade = (totalFemales > 0) ? (double)totalFemaleGrades / totalFemales : 0.0;
    cout << averageFemaleGrade << endl;

    return 0;
}
```

Date- 3 May S1

Q1- The Organization has data warehouse there will be given a three digit number. Check Whether number is divisible by 9 or not ?

I/P- 162

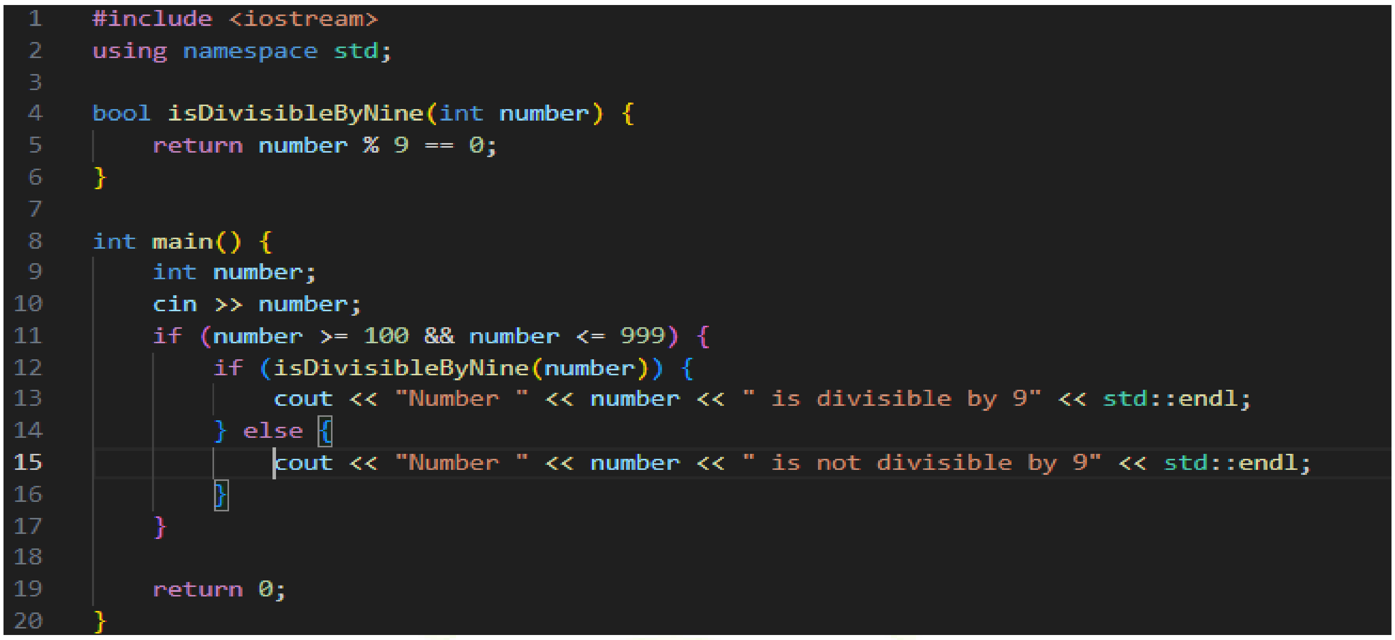
O/P- Number 162 is divisible by 9.

I/P 236

O/P- Number 236 is not divisible by 9.

Approach-

In this approach, the function isDivisibleByNine checks if a given number is divisible by 9 by using the modulus operator. In the main function, it reads an integer input and first checks if the number is within the range of 100 to 999. If it is, the function isDivisibleByNine is called to determine divisibility. Based on the result, the program prints whether the number is divisible by 9 or not.



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

bool isDivisibleByNine(int number) {
    return number % 9 == 0;
}

int main() {
    int number;
    cin >> number;
    if (number >= 100 && number <= 999) {
        if (isDivisibleByNine(number)) {
            cout << "Number " << number << " is divisible by 9" << std::endl;
        } else {
            cout << "Number " << number << " is not divisible by 9" << std::endl;
        }
    }

    return 0;
}
```

Q2- We are given a list of numbers we need to return maximum difference b/w smallest & largest Number.

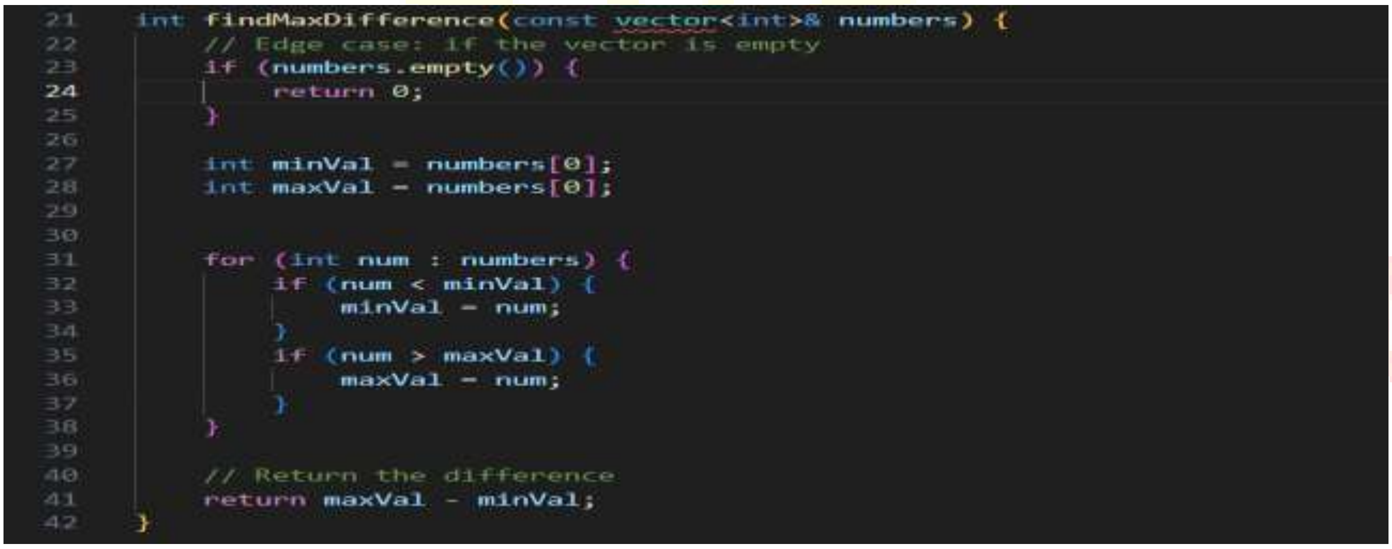
I/P- n=4

[1,5,7,3]

O/P- 6

Approach-

In this approach, the function findMaxDifference calculates the maximum difference between the smallest and largest numbers in a vector. It starts by initializing the minVal and maxVal to the first element of the vector. As it iterates through each element, it updates minVal and maxVal if a smaller or larger value is found. Finally, it returns the difference between the largest and smallest values.



```cpp
int findMaxDifference(const vector<int>& numbers) {
    // Edge case: if the vector is empty
    if (numbers.empty()) {
        return 0;
    }

    int minVal = numbers[0];
    int maxVal = numbers[0];


    for (int num : numbers) {
        if (num < minVal) {
            minVal = num;
        }
        if (num > maxVal) {
            maxVal = num;
        }
    }

    // Return the difference
    return maxVal - minVal;
}
```

Date- 3May S2

Q1-A person has many shoes of different sizes and he wants to arrange them, Calulate the numbers of pairs of shoes.

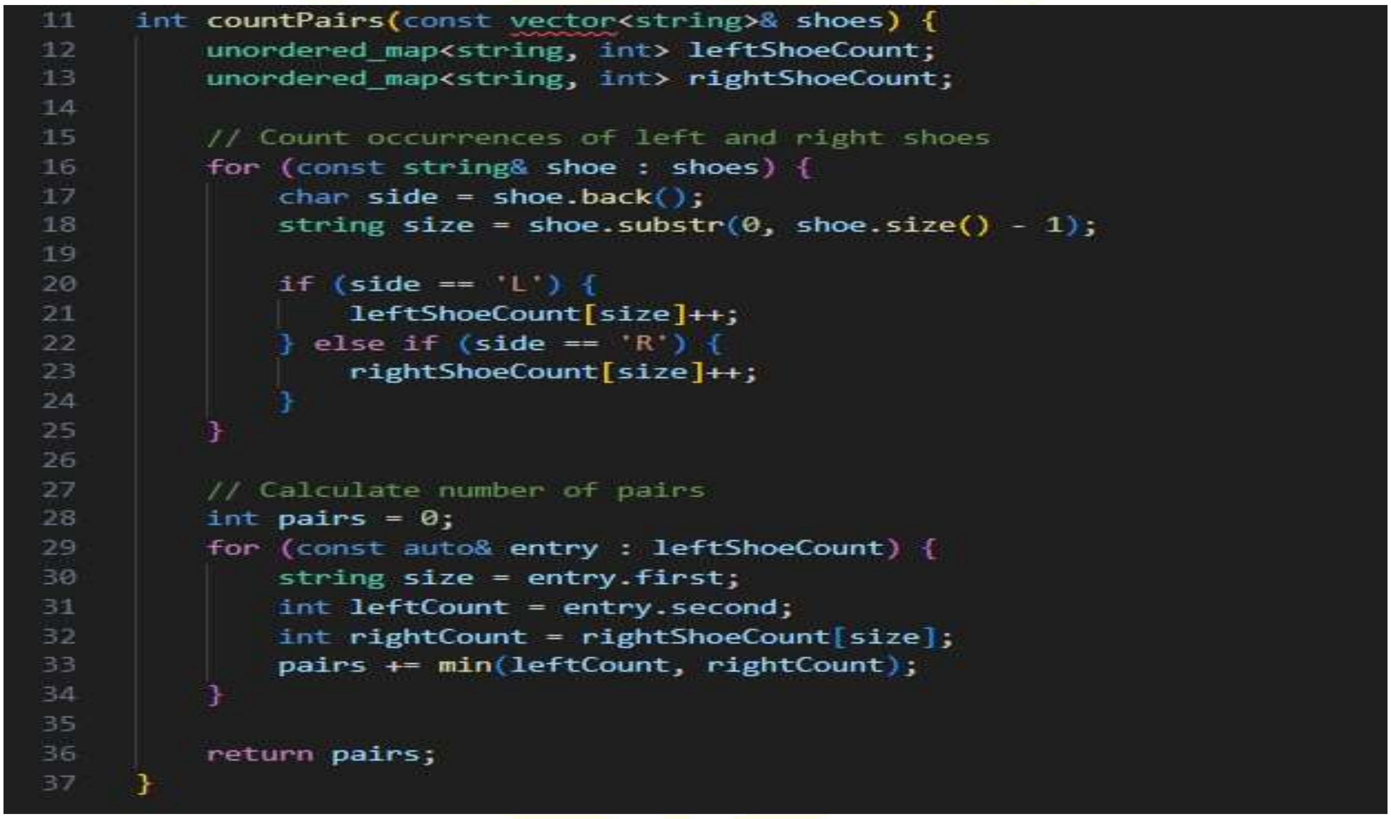
I/P- 6

7L 8L 6R 6L 7L 8R

O/P- 2

Approach-

In this approach, the function `countPairs` calculates the number of shoe pairs from a list of shoes. It uses two unordered maps to count the occurrences of left and right shoes for each size. The function then iterates over the map of left shoes, and for each size, it finds the corresponding count of right shoes. It adds the minimum of the counts for each size to the total number of pairs. Finally, it returns the total number of pairs.



```cpp
int countPairs(const vector<string>& shoes) {
    unordered_map<string, int> leftShoeCount;
    unordered_map<string, int> rightShoeCount;

    // Count occurrences of left and right shoes
    for (const string& shoe : shoes) {
        char side = shoe.back();
        string size = shoe.substr(0, shoe.size() - 1);

        if (side == 'L') {
            leftShoeCount[size]++;
        } else if (side == 'R') {
            rightShoeCount[size]++;
        }
    }

    // Calculate number of pairs
    int pairs = 0;
    for (const auto& entry : leftShoeCount) {
        string size = entry.first;
        int leftCount = entry.second;
        int rightCount = rightShoeCount[size];
        pairs += min(leftCount, rightCount);
    }

    return pairs;
}
```

Q2- In a company there are employees and their efficiency is given in array (can be negative) you need to find the maximum efficiency of 3 employees. The efficiency of 3 employees will be calculated by multiplying their individual efficiencies from the given array.

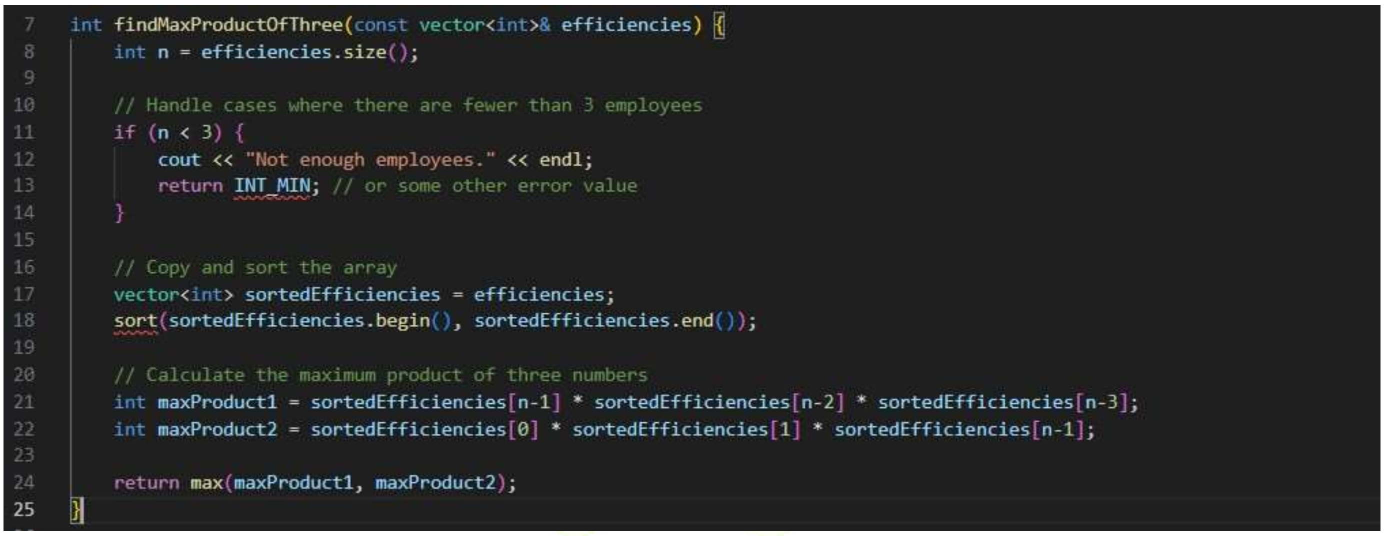
I/P - 5

[2,-5,7,6,3]

O/P – 84

Approach-

In this approach, the function `findMaxProductOfThree` computes the maximum product of any three numbers from a vector of efficiencies. It first checks if there are at least three employees; if not, it returns an error value. The function then sorts the vector to facilitate easy access to the largest and smallest values. It calculates the maximum product of the top three largest numbers and the product of the two smallest (potentially negative) numbers with the largest number. The function returns the greater of these two products as the maximum product of three numbers.



```cpp
int findMaxProductOfThree(const vector<int>& efficiencies) {
    int n = efficiencies.size();

    // Handle cases where there are fewer than 3 employees
    if (n < 3) {
        cout << "Not enough employees." << endl;
        return INT_MIN; // or some other error value
    }

    // Copy and sort the array
    vector<int> sortedEfficiencies = efficiencies;
    sort(sortedEfficiencies.begin(), sortedEfficiencies.end());

    // Calculate the maximum product of three numbers
    int maxProduct1 = sortedEfficiencies[n-1] * sortedEfficiencies[n-2] * sortedEfficiencies[n-3];
    int maxProduct2 = sortedEfficiencies[0] * sortedEfficiencies[1] * sortedEfficiencies[n-1];

    return max(maxProduct1, maxProduct2);
}
```

Date- 6 May S1

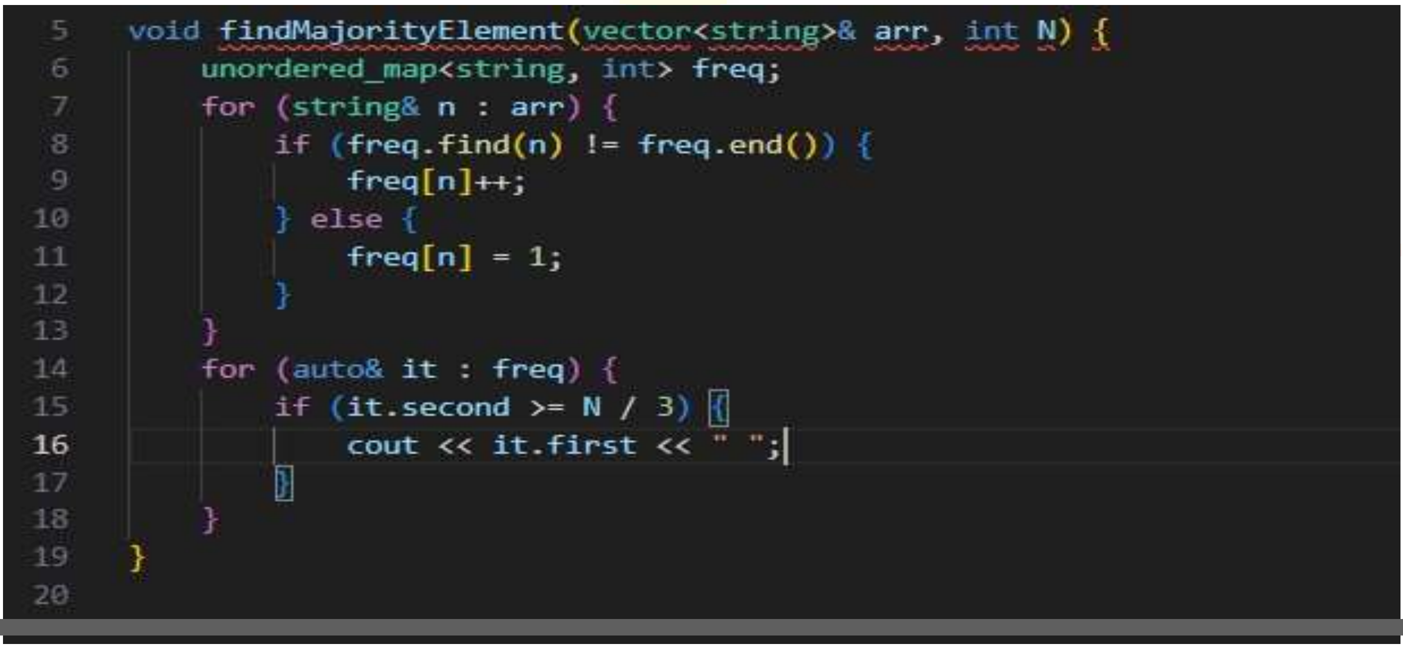
Q1- Given an array nums of size n, return the majority element.The Majority elements is the element that appears more than n/3 times.You may assume that the majority element always exists in the array.

I/P- nums=[2,2,2,4,5,5]

O/P-2

Approach-

In this approach, the function `findMajorityElement` identifies and prints elements in the vector that appear at least (N/3) times. It uses an unordered map to count the frequency of each element. After populating the map with counts, it iterates through the map and prints elements whose frequency meets or exceeds the (N/3) threshold. This approach efficiently determines majority elements based on their occurrence in the vector.



```cpp
void findMajorityElement(vector<string>& arr, int N) {
    unordered_map<string, int> freq;
    for (string& n : arr) {
        if (freq.find(n) != freq.end()) {
            freq[n]++;
        } else {
            freq[n] = 1;
        }
    }
    for (auto& it : freq) {
        if (it.second >= N / 3) {
            cout << it.first << " ";
        }
    }
}
```

Q2: Given an array balls containing numbers like '3', '5', and '6', sort the array in-place so that balls with the same number

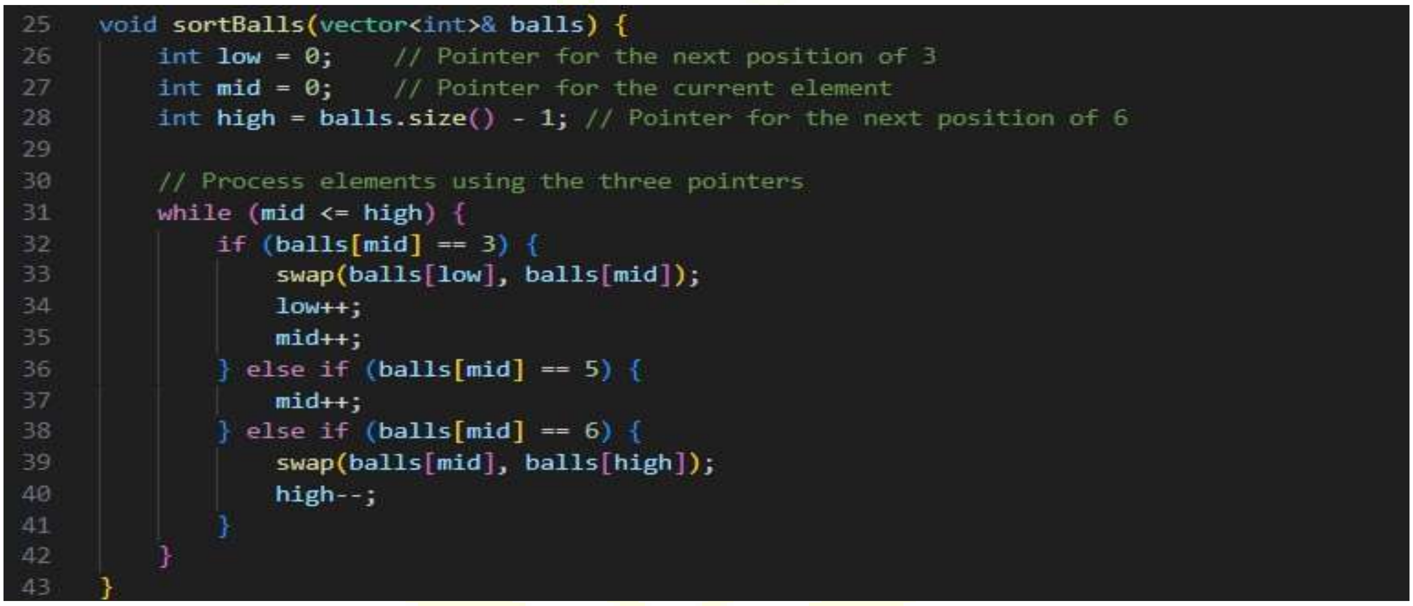
are adjacent, in the order '3', '5', '6'.

I/P- balls = [5, 3, 3, 6, 5, 6]

O/P- [3, 3, 5, 5, 6, 6]

Approach-

In this approach, the function sortBalls sorts an array of balls with values 3, 5, and 6 using the Dutch National Flag algorithm. Three pointers (low, mid, and high) are used to manage the positions of these values in the array. The low pointer tracks where the next 3 should be placed, the mid pointer iterates through the array, and the high pointer tracks where the next 6 should be placed. The algorithm swaps values based on their type to ensure that all 3s are at the beginning, 5s in the middle, and 6s at the end of the array.



```
void sortBalls(vector<int>& balls) {
    int low = 0;    // Pointer for the next position of 3
    int mid = 0;    // Pointer for the current element
    int high = balls.size() - 1; // Pointer for the next position of 6

    // Process elements using the three pointers
    while (mid <= high) {
        if (balls[mid] == 3) {
            swap(balls[low], balls[mid]);
            low++;
            mid++;
        } else if (balls[mid] == 5) {
            mid++;
        } else if (balls[mid] == 6) {
            swap(balls[mid], balls[high]);
            high--;
        }
    }
}
```

Date- 6 May S2

Q1- Given an array of size N-1 with integers in the range of [1,N] the task to find the missing number from the first N integers.

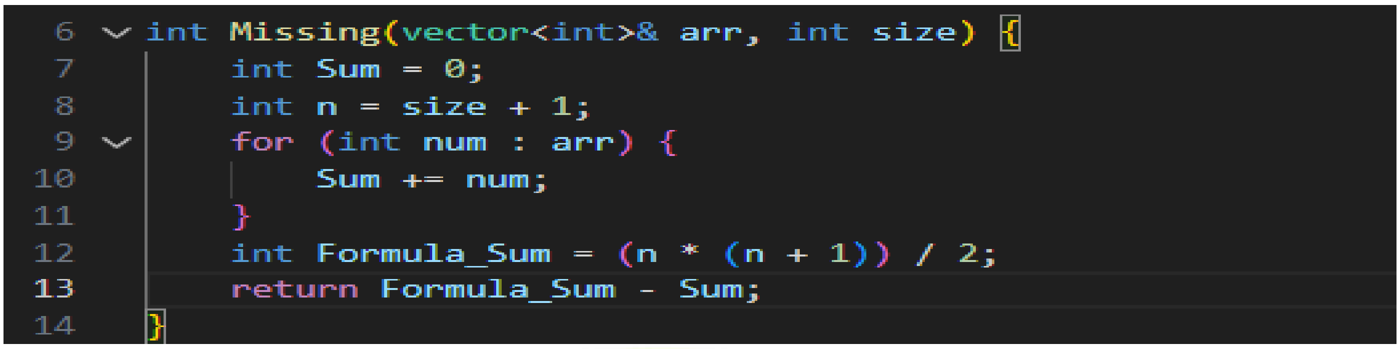
I/P- 6

[1,2,3,4,6,7]

O/P- 5

Approach-

In this approach, the function `Missing` finds the missing number in a sequence from 1 to (n). It calculates the sum of all elements in the given array and the expected sum of the first (n) natural numbers using the formula n(n + 1)/2. The missing number is then determined by subtracting the actual sum from the expected sum. This method efficiently identifies the missing number with a time complexity of (O(n)) and a space complexity of (O(1)).



```cpp
int Missing(vector<int>& arr, int size) {
    int Sum = 0;
    int n = size + 1;
    for (int num : arr) {
        Sum += num;
    }
    int Formula_Sum = (n * (n + 1)) / 2;
    return Formula_Sum - Sum;
}
```

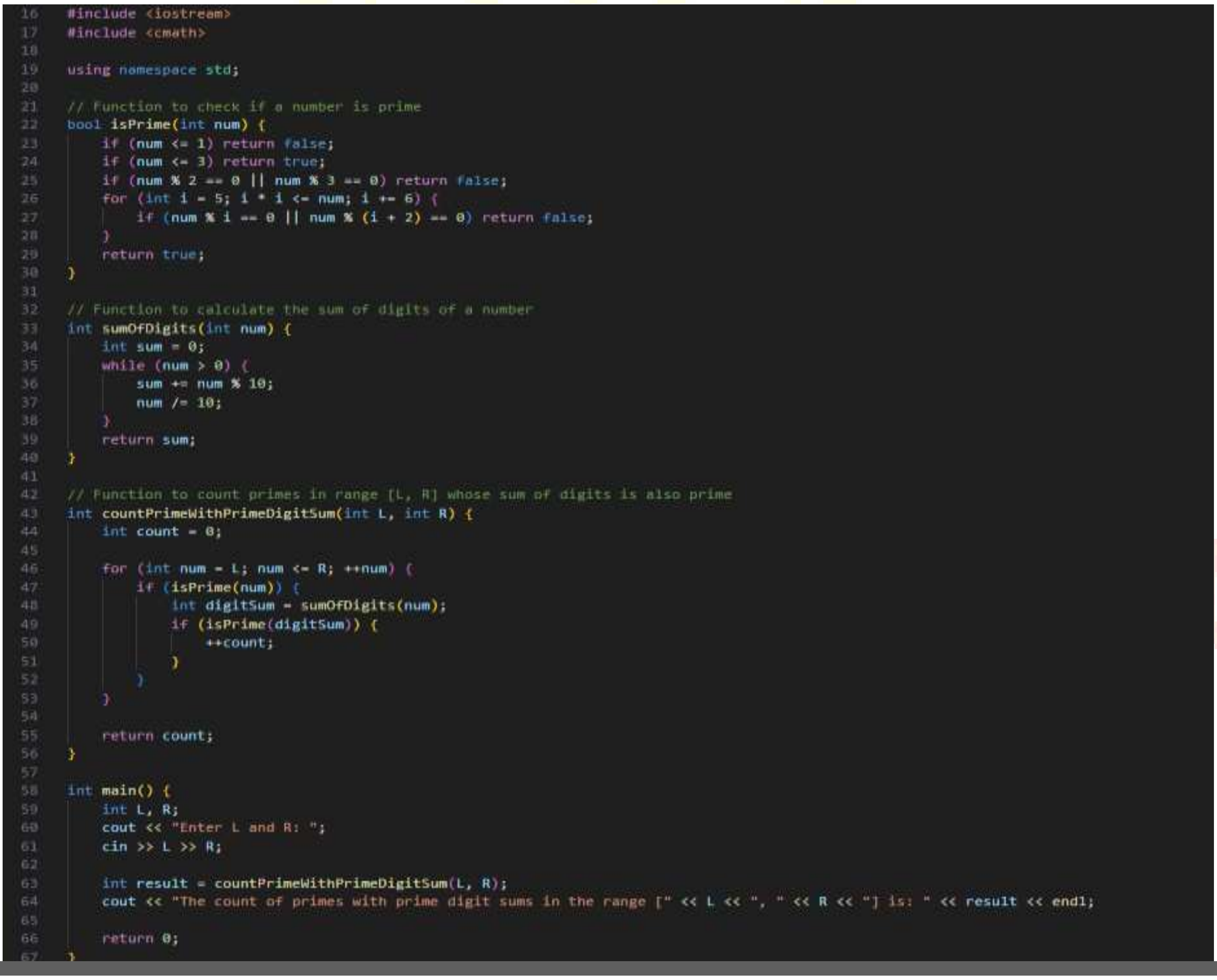
Q2-Given two integers L and R, the task is to find the count of total numbers of prime numbers in the range [L, R] whose sum of the digits is also a prime number.

I/P- L=1,R=10

O/P- 4

Approach-

In this approach, the function `countPrimeWithPrimeDigitSum` identifies prime numbers within a given range `[L, R]` where the sum of their digits is also prime. It uses the `isPrime` function to check for primality of both the number and its digit sum. The `sumOfDigits` function computes the sum of digits for each number. The total count of such numbers is returned, providing a count of primes with a prime digit sum in the specified range.



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

using namespace std;

// Function to check if a number is prime
bool isPrime(int num) {
    if (num <= 1) return false;
    if (num <= 3) return true;
    if (num % 2 == 0 || num % 3 == 0) return false;
    for (int i = 5; i * i <= num; i += 6) {
        if (num % i == 0 || num % (i + 2) == 0) return false;
    }
    return true;
}

// Function to calculate the sum of digits of a number
int sumOfDigits(int num) {
    int sum = 0;
    while (num > 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

// Function to count primes in range [L, R] whose sum of digits is also prime
int countPrimeWithPrimeDigitSum(int L, int R) {
    int count = 0;

    for (int num = L; num <= R; ++num) {
        if (isPrime(num)) {
            int digitSum = sumOfDigits(num);
            if (isPrime(digitSum)) {
                ++count;
            }
        }
    }

    return count;
}

int main() {
    int L, R;
    cout << "Enter L and R: ";
    cin >> L >> R;

    int result = countPrimeWithPrimeDigitSum(L, R);
    cout << "The count of primes with prime digit sums in the range [" << L << ", " << R << "] is: " << result << endl;

    return 0;
}
```

Date- 8May S1

Q1- Write a program to take input of X and Y in a new line.Print the number which is nearer to the integer when divided by Y.

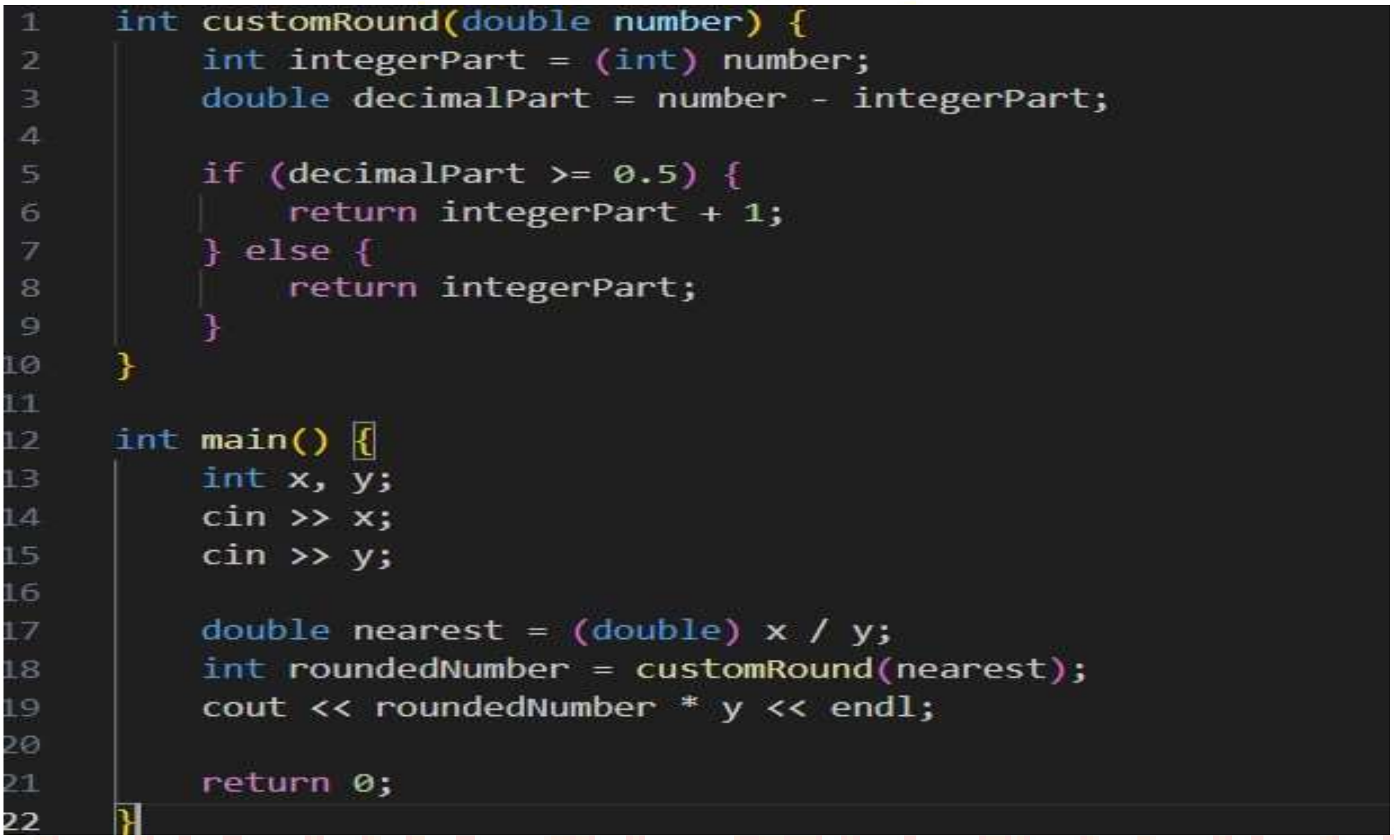
I/P- X=25

Y=4

O/P-24

Approach –

In this approach, the program first defines a custom rounding function, customRound, that rounds a given double to the nearest integer, rounding up if the decimal part is 0.5 or greater. In the main function, the user inputs two integers x and y. The program calculates the quotient of x divided by y, rounds it using the customRound function, and then multiplies the rounded value by y to get the nearest multiple of y to x, which is then printed



```
int customRound(double number) {
    int integerPart = (int) number;
    double decimalPart = number - integerPart;

    if (decimalPart >= 0.5) {
        return integerPart + 1;
    } else {
        return integerPart;
    }
}

int main() {
    int x, y;
    cin >> x;
    cin >> y;

    double nearest = (double) x / y;
    int roundedNumber = customRound(nearest);
    cout << roundedNumber * y << endl;

    return 0;
}
```

Q2-Write a program that generates a password adhering to the following conditions:

C1-The password must consist of at least 8 characters.

C2-It must contain at least one integer.

C3-It must contain at least one special character from the set {'#', '@').

C4-It must contain at least one uppercase letter and one lowercase letter.

C5-Each character in the password should be incremented by the number of times specified by the second input.

Your program should take two inputs:

1. A string representing the initial password.

2. An integer representing the number of times each character should be incremented.

Your program should then generate and output the modified password.

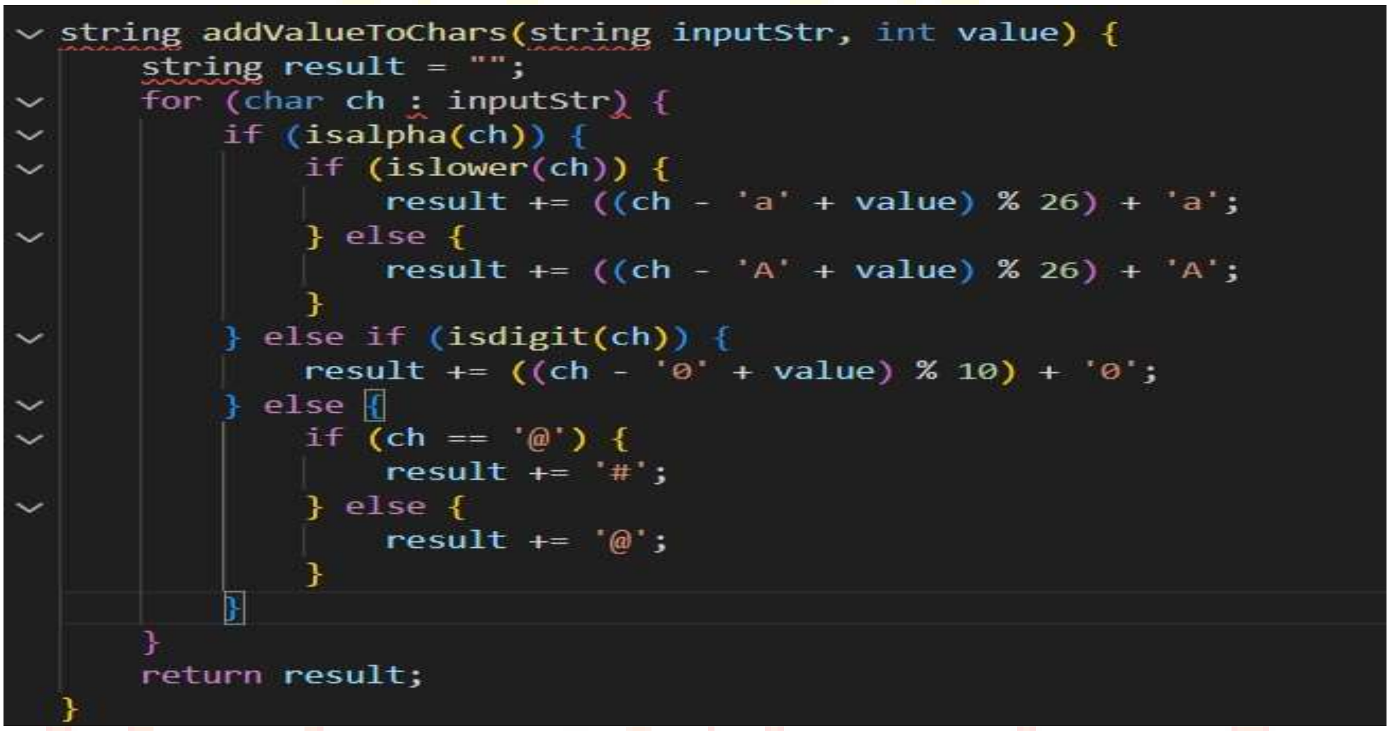
I/P 1: werV432@

I/P 2: 2

O/P: ygtX653#

Approach-

In this approach, the function addValueToChars processes each character of the input string. If the character is a letter, it shifts it by a given value within the alphabet, wrapping around using modulo arithmetic. If the character is a digit, it shifts similarly within the range of digits. Non-alphanumeric characters like @ are conditionally transformed into # and vice versa. The modified characters are then concatenated to form the resulting string, which is returned.



```
string addValueToChars(string inputStr, int value) {
    string result = "";
    for (char ch : inputStr) {
        if (isalpha(ch)) {
            if (islower(ch)) {
                result += ((ch - 'a' + value) % 26) + 'a';
            } else {
                result += ((ch - 'A' + value) % 26) + 'A';
            }
        } else if (isdigit(ch)) {
            result += ((ch - '0' + value) % 10) + '0';
        } else {
            if (ch == '@') {
                result += '#';
            } else {
                result += '@';
            }
        }
    }
    return result;
}
```

Date- 8 May S2

Q1-Given size of n and list of array elements and we should print if the

given element in array is divisible by 3 then replace the element with "Three"

and if the element in array is divisible by 5 then replace the element with "Five"

if the element divisible by 3 and 5 both then replace the element with "ThreeFive"

if the element in the array is not satisfying the above 3 conditions then

put the element as it is and print the array

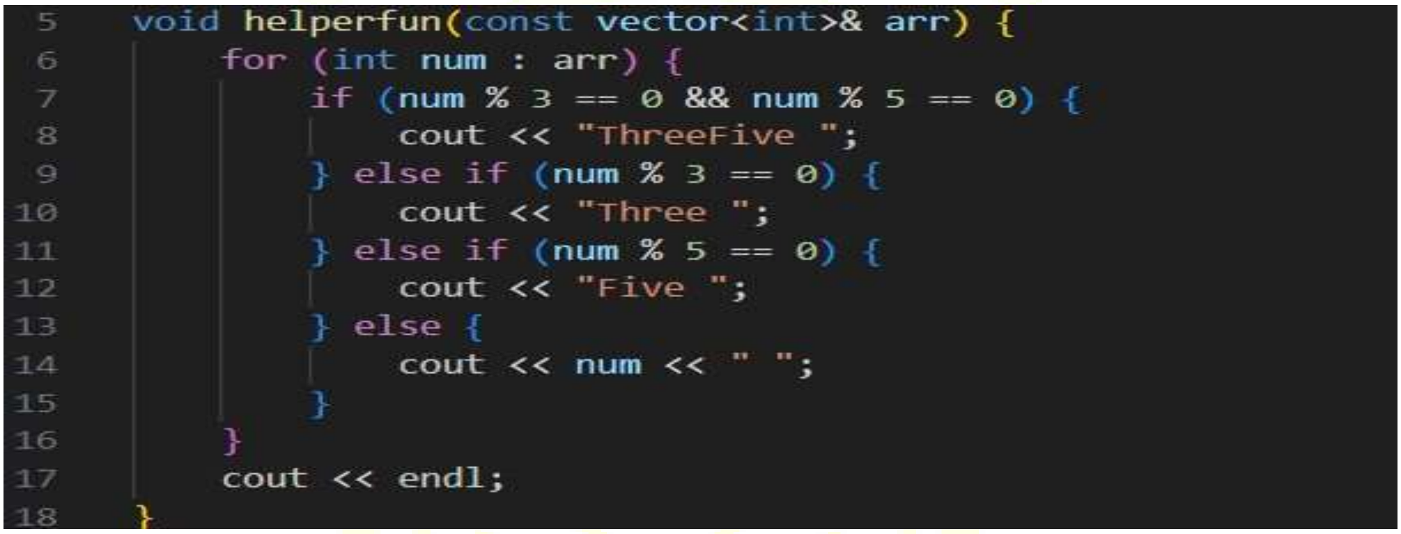
I/P- N=4

[8,3,9,5]

O/P- 8 Three 9 Five

Approach-

In this approach, the function `helperfun` iterates through the given array and checks each element to determine if it is divisible by 3, 5, or both. If a number is divisible by both 3 and 5, it prints "ThreeFive"; if only by 3, it prints "Three"; and if only by 5, it prints "Five". If the number is divisible by neither, it prints the number itself. This is similar to the FizzBuzz problem, where numbers are replaced by strings based on their divisibility.



```
void helperfun(const vector<int>& arr) {
    for (int num : arr) {
        if (num % 3 == 0 && num % 5 == 0) {
            cout << "ThreeFive ";
        } else if (num % 3 == 0) {
            cout << "Three ";
        } else if (num % 5 == 0) {
            cout << "Five ";
        } else {
            cout << num << " ";
        }
    }
    cout << endl;
}
```

Q2- Task 1- print the collatz sequence upto ending with 1 the sequence should be in the following way

-> if the number is even:

F(n)=n//2

-> if the number is odd:

F(n)=3*n+1

This sequence should end until the last element of sequence is 1

Task-2

For the given integer from 1,n it should calculate the sequence of each k value Le, 1<=k<=n

Calculate the maximum length of sequence list of each k value and return the maximum length of the sequence list of the k value and the k value itself

Task-3

For the given integer from 1.n it should calculate the sequence of each k value le, 1<=k<=n Calculate the maximum value of the each sequence within the sequence list of

each k value and return the maximum value of the sequence list of the k value and those k value itself

Input: 5

Output:

[5,16,8,4,2.1]

8,3

16,3

Input: 0

Output: Error!

Input: xyza

output: Error!

Input:-13

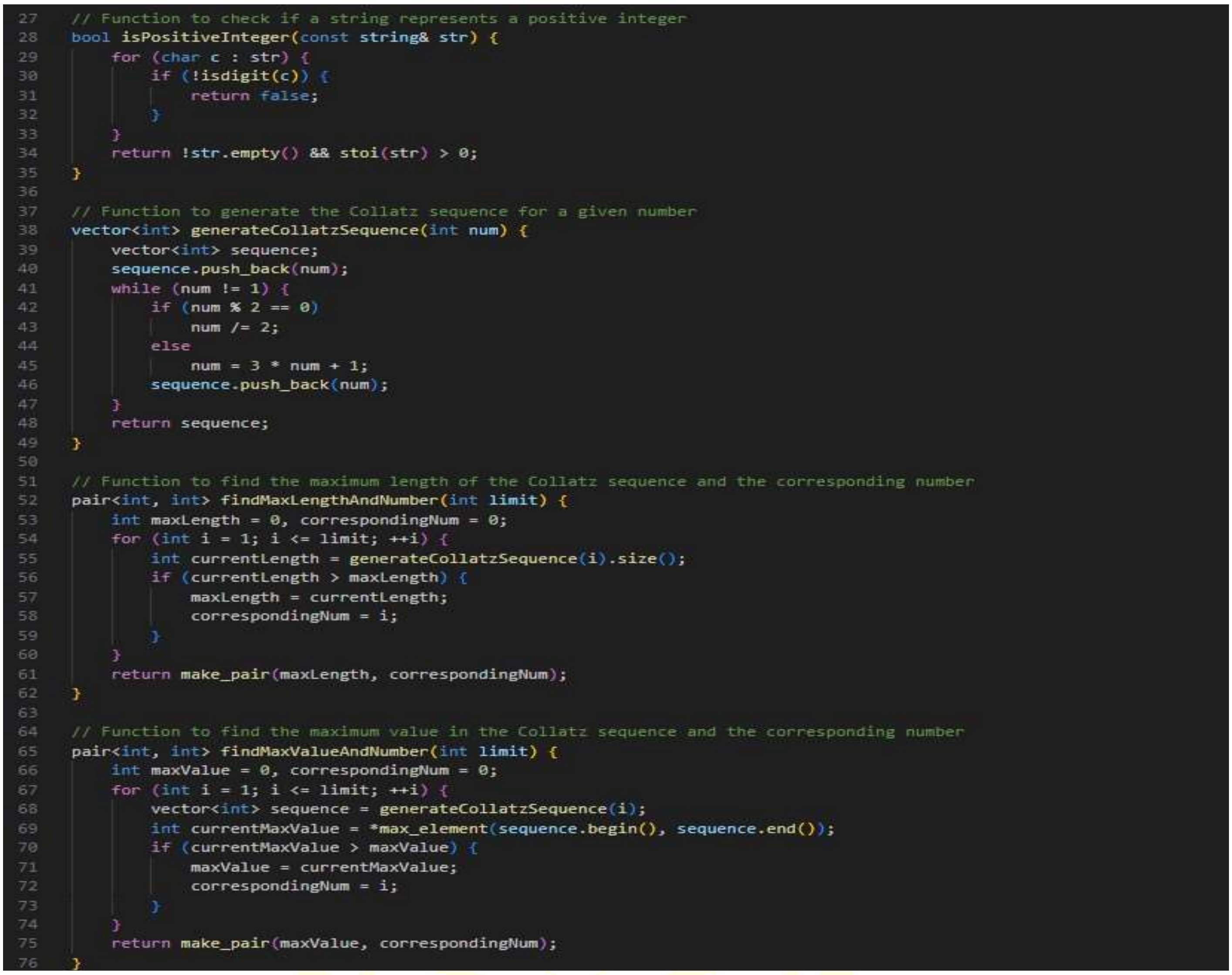
Output: Error

Input: 5.5

Output: Error!

Approach –

In this approach, the code starts by checking if the input string represents a positive integer. If valid, it generates the Collatz sequence starting from the given integer n, where the sequence follows the rule: divide by 2 if even, or multiply by 3 and add 1 if odd, until n reaches 1. It then finds the number k between 1 and n that produces the longest sequence and the highest value within any sequence. The results include the entire sequence for n, the length of the longest sequence, and the highest value encountered. If the input is not valid, an error message is displayed.



```cpp
// Function to check if a string represents a positive integer
bool isPositiveInteger(const string& str) {
    for (char c : str) {
        if (!isdigit(c)) {
            return false;
        }
    }
    return !str.empty() && stoi(str) > 0;
}

// Function to generate the Collatz sequence for a given number
vector<int> generateCollatzSequence(int num) {
    vector<int> sequence;
    sequence.push_back(num);
    while (num != 1) {
        if (num % 2 == 0)
            num /= 2;
        else
            num = 3 * num + 1;
        sequence.push_back(num);
    }
    return sequence;
}

// Function to find the maximum length of the Collatz sequence and the corresponding number
pair<int, int> findMaxLengthAndNumber(int limit) {
    int maxLength = 0, correspondingNum = 0;
    for (int i = 1; i <= limit; ++i) {
        int currentLength = generateCollatzSequence(i).size();
        if (currentLength > maxLength) {
            maxLength = currentLength;
            correspondingNum = i;
        }
    }
    return make_pair(maxLength, correspondingNum);
}

// Function to find the maximum value in the Collatz sequence and the corresponding number
pair<int, int> findMaxValueAndNumber(int limit) {
    int maxValue = 0, correspondingNum = 0;
    for (int i = 1; i <= limit; ++i) {
        vector<int> sequence = generateCollatzSequence(i);
        int currentMaxValue = *max_element(sequence.begin(), sequence.end());
        if (currentMaxValue > maxValue) {
            maxValue = currentMaxValue;
            correspondingNum = i;
        }
    }
    return make_pair(maxValue, correspondingNum);
}
```

Date- 9May S1

Q1) given an integer n, return true if it is an armstrong number, otherwise return false.

an armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

input:

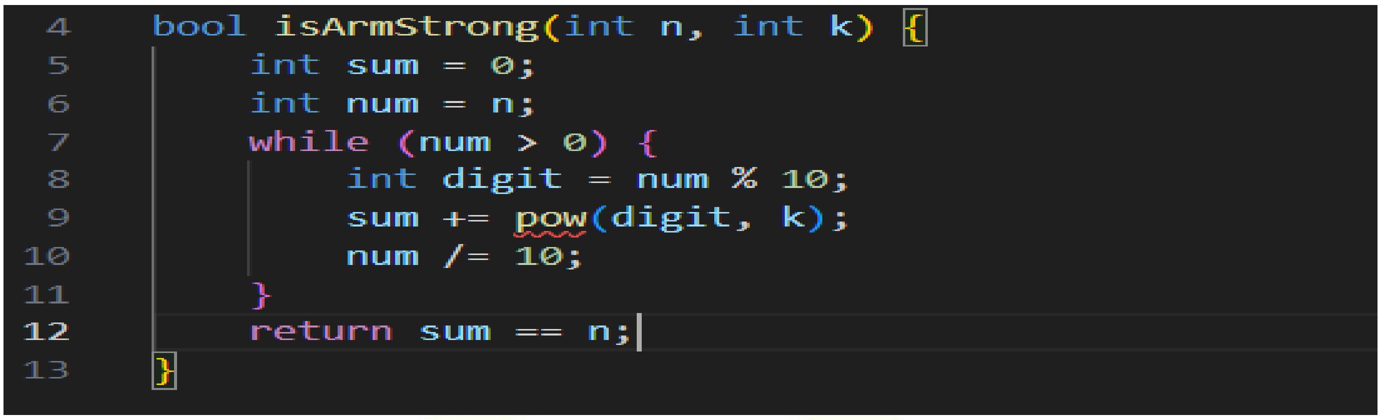
[153, 371, 108]

output:

[true, true, false]

Approach-

In this approach, the function `isArmStrong` determines if a number `n` is an Armstrong number in `k`-digit space. An Armstrong number (or Narcissistic number) is one where the sum of its digits each raised to the power of `k` equals the number itself. The function computes this by iterating over each digit of `n`, raising it to the power of `k`, and summing these values. Finally, it checks if this sum matches the original number, returning `true` if they are equal and `false` otherwise.



```cpp
bool isArmStrong(int n, int k) {
    int sum = 0;
    int num = n;
    while (num > 0) {
        int digit = num % 10;
        sum += pow(digit, k);
        num /= 10;
    }
    return sum == n;
}
```

Q2)Print Top Candidate from the given input

INPUT format: no. of input n ,top k to be sorted, candidate with marks

I/P:3 4 70 Johnny 85 Anmol 270 Vishy

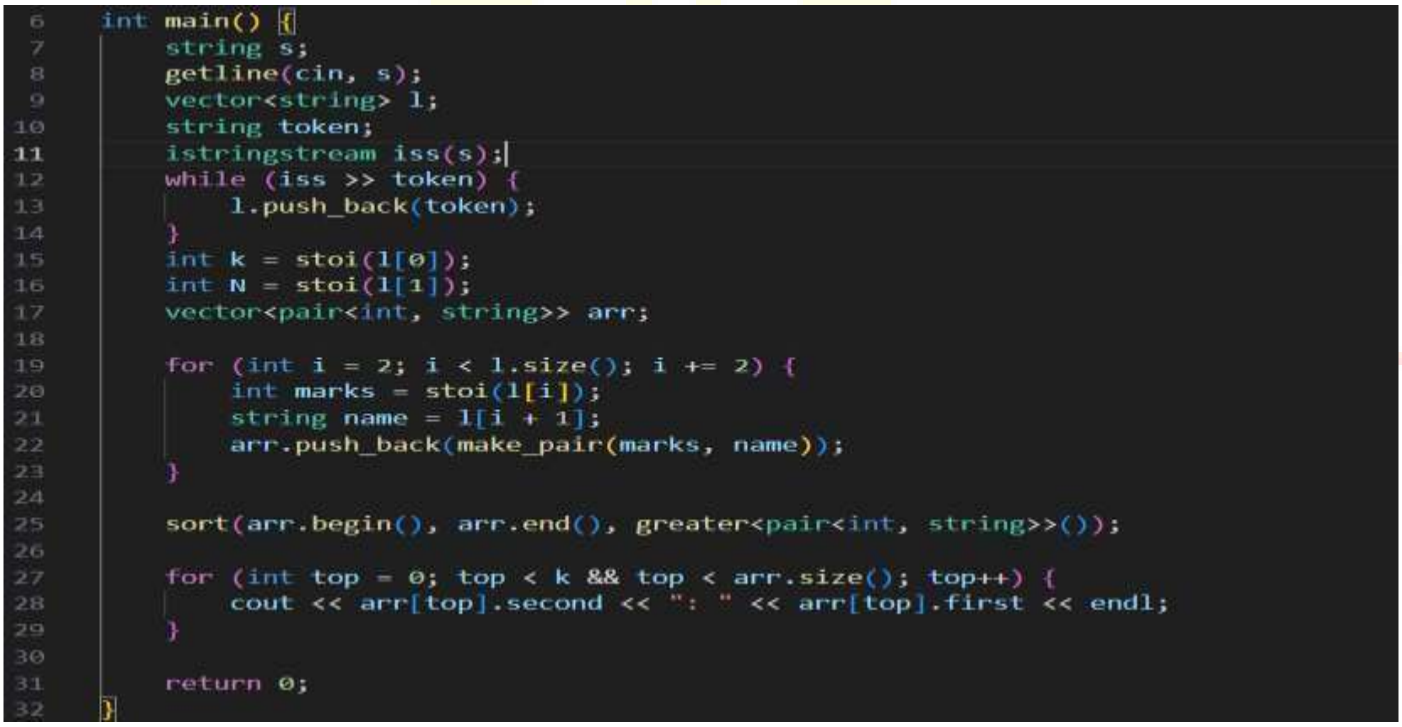
O/P:Vishy:270

Anmol:85

Jhonny 70

Approach-

In this approach, the `main` function reads a line of input, which includes an integer `k`, an integer `N`, followed by pairs of `marks` and `name`. The input is split into tokens using `istringstream` and parsed into a vector `l`. The first two tokens are converted to integers `k` and `N`, respectively. Subsequent tokens are processed into pairs of integers (marks) and strings (names), which are stored in a vector of pairs. This vector is then sorted in descending order based on marks using `sort` with a custom comparator. Finally, the top `k` entries (or fewer if there aren't `k` entries) are printed, showing the names and their corresponding marks.



```cpp
int main() {
    string s;
    getline(cin, s);
    vector<string> l;
    string token;
    istringstream iss(s);
    while (iss >> token) {
        l.push_back(token);
    }
    int k = stoi(l[0]);
    int N = stoi(l[1]);
    vector<pair<int, string>> arr;

    for (int i = 2; i < l.size(); i += 2) {
        int marks = stoi(l[i]);
        string name = l[i + 1];
        arr.push_back(make_pair(marks, name));
    }

    sort(arr.begin(), arr.end(), greater<pair<int, string>>());

    for (int top = 0; top < k && top < arr.size(); top++) {
        cout << arr[top].second << ": " << arr[top].first << endl;
    }

    return 0;
}
```

Date-9 May S2

Q1)Print all combinations

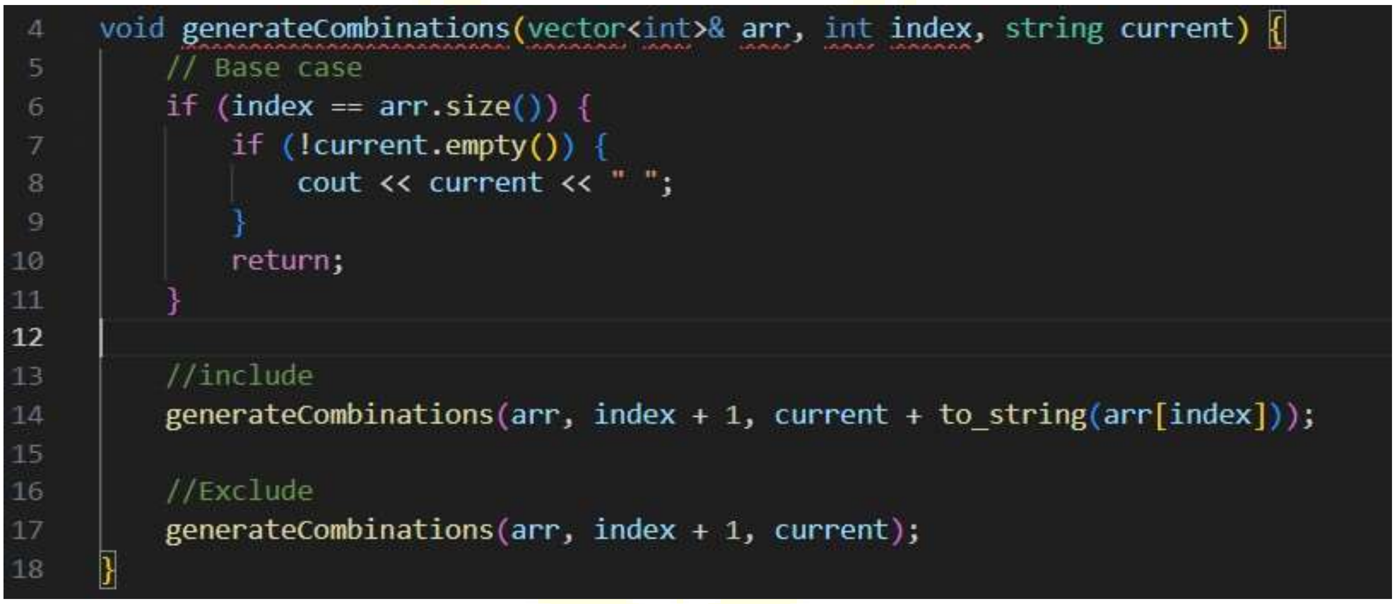
I/P- n=3

[0,1,2]

O/P-[0,01,012,0123,1,12,123,2,23,3]

Approach –

In this approach, the `generateCombinations` function recursively generates all possible combinations of elements from the given vector `arr`. It starts at the specified `index` and uses the `current` string to build combinations. For each element at `arr[index]`, the function is called twice: once including the current element in the combination (appending it to `current`), and once excluding it (not modifying `current`). The base case is when `index` reaches the end of the vector, at which point it prints the current combination if it's not empty. This recursive method explores all subsets of the array, including the empty subset.



```
void generateCombinations(vector<int>& arr, int index, string current) {
    // Base case
    if (index == arr.size()) {
        if (!current.empty()) {
            cout << current << " ";
        }
        return;
    }

    //include
    generateCombinations(arr, index + 1, current + to_string(arr[index]));

    //Exclude
    generateCombinations(arr, index + 1, current);
}
```

Date-14May S1

Q1)You are given an integer array num. The unique elements of an array are the elements that appear exactly once in the array.

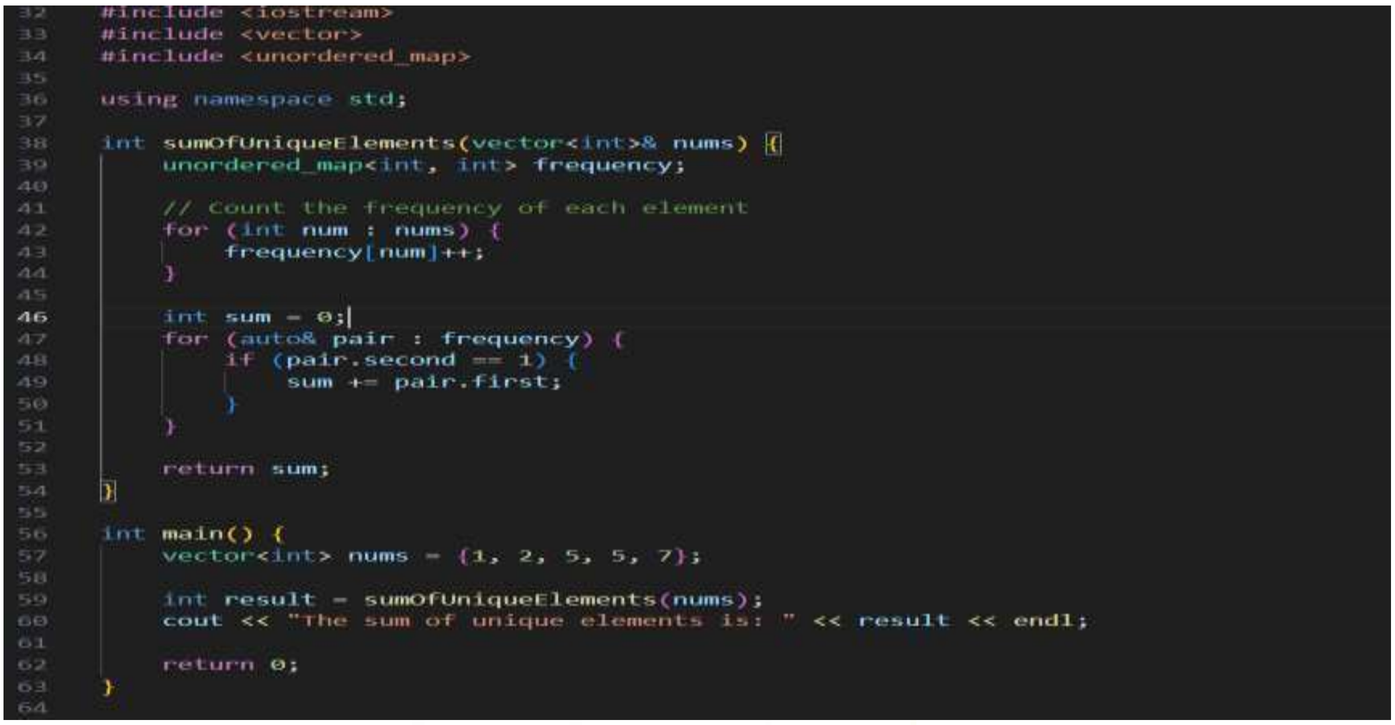
Return sum of all unique elements of nums.

I/P- [1,2,5,5,7]

O/P-10

Approach –

In this approach, the sumOfUniqueElements function calculates the sum of all unique elements in a vector nums. It first uses an unordered_map to count the frequency of each element. Then, it iterates through this map, adding elements to the sum only if they appear exactly once in the array. The final sum of these unique elements is returned.



```
#include <iostream>
#include <vector>
#include <unordered_map>

using namespace std;

int sumOfUniqueElements(vector<int>& nums) {
    unordered_map<int, int> frequency;

    // Count the frequency of each element
    for (int num : nums) {
        frequency[num]++;
    }

    int sum = 0;
    for (auto& pair : frequency) {
        if (pair.second == 1) {
            sum += pair.first;
        }
    }

    return sum;
}

int main() {
    vector<int> nums = {1, 2, 5, 5, 7};

    int result = sumOfUniqueElements(nums);
    cout << "The sum of unique elements is: " << result << endl;

    return 0;
}
```

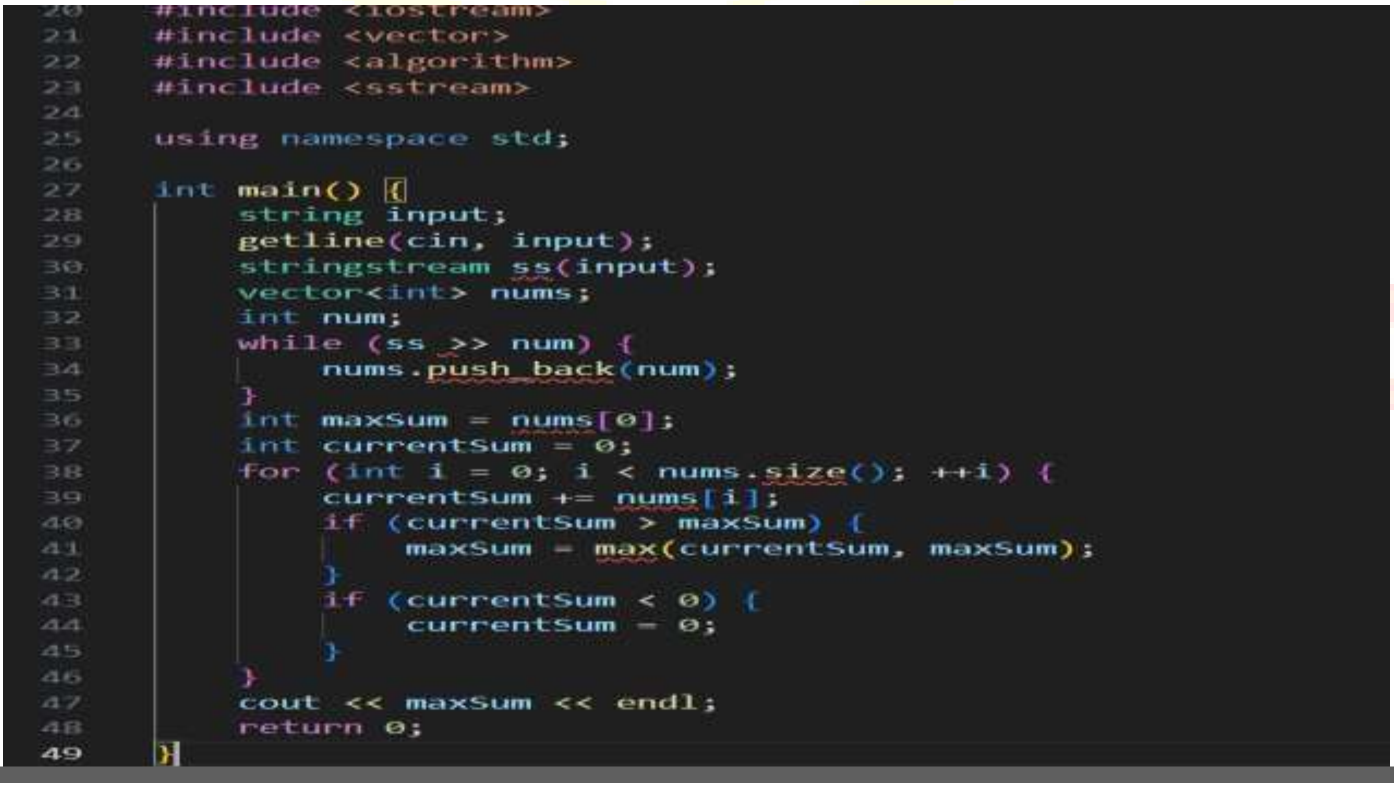
Q2)Given an integer array nums ,find the subarray with largest sum , and return its sum.

I/P-[-1,2,4,2,-5,3]

O/P-8

Approach-

In this approach, the program reads a string of space-separated integers from the input and stores them in a vector. It then calculates the maximum sum of any contiguous subarray using a modified version of Kadane's algorithm. The currentSum keeps track of the sum of the current subarray, updating the maxSum whenever a higher sum is found. If currentSum becomes negative, it is reset to zero. Finally, the maximum subarray sum is printed.



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

using namespace std;

int main() {
    string input;
    getline(cin, input);
    stringstream ss(input);
    vector<int> nums;
    int num;
    while (ss >> num) {
        nums.push_back(num);
    }
    int maxSum = nums[0];
    int currentSum = 0;
    for (int i = 0; i < nums.size(); ++i) {
        currentSum += nums[i];
        if (currentSum > maxSum) {
            maxSum = max(currentSum, maxSum);
        }
        if (currentSum < 0) {
            currentSum = 0;
        }
    }
    cout << maxSum << endl;
    return 0;
}
```

Date- 14 May S2

Q1)You are tasked with writing a program to calculate the total shipping cost based on the weight of the package and the distance it needs to travel. The shipping cost is determined by the following criteria:

1. Base money: $5.00

2. Cost per kilogram: $2.00

3. Cost per 10 kilometers: $0.5

Examplel:

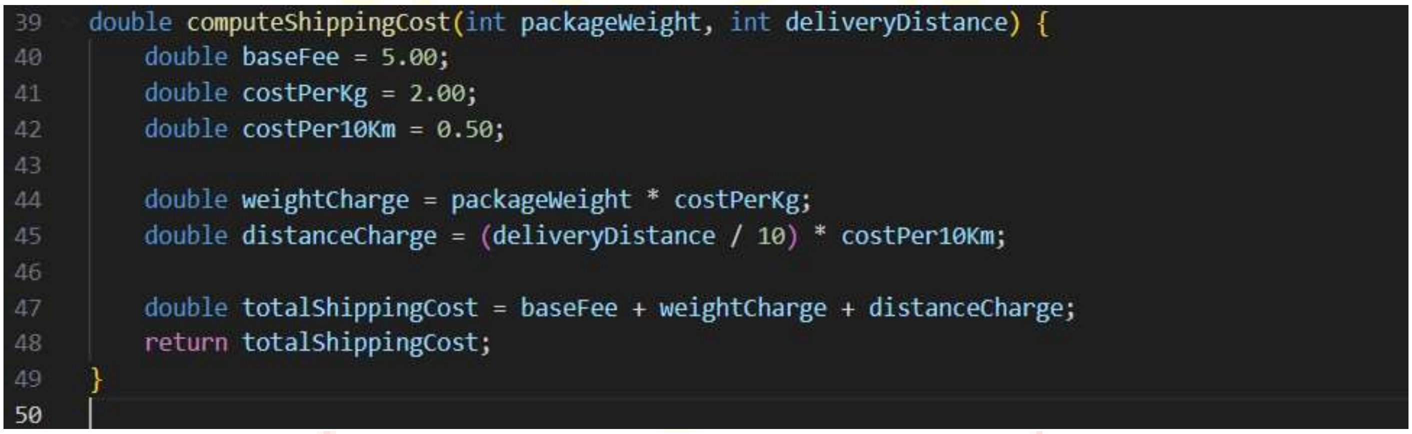
10 (w)

100 (D)

output: $30.00

Approach –

In this approach, the computeShippingCost function calculates the total shipping cost for a package based on its weight and delivery distance. It starts with a base fee of $5.00. The cost is then increased based on the weight of the package, where each kilogram costs $2.00. Additionally, the cost is incremented based on the delivery distance, with every 10 kilometers adding $0.50. The total shipping cost is the sum of these charges.



```
double computeShippingCost(int packageWeight, int deliveryDistance) {
    double baseFee = 5.00;
    double costPerKg = 2.00;
    double costPer10Km = 0.50;

    double weightCharge = packageWeight * costPerKg;
    double distanceCharge = (deliveryDistance / 10) * costPer10Km;

    double totalShippingCost = baseFee + weightCharge + distanceCharge;
    return totalShippingCost;
}
```

Q2) Given an array of integers nums and an integer k, return the tota number of subarrays whose sum equals to k. A subarray is a contiguous non-empty sequence of elements withi an array.

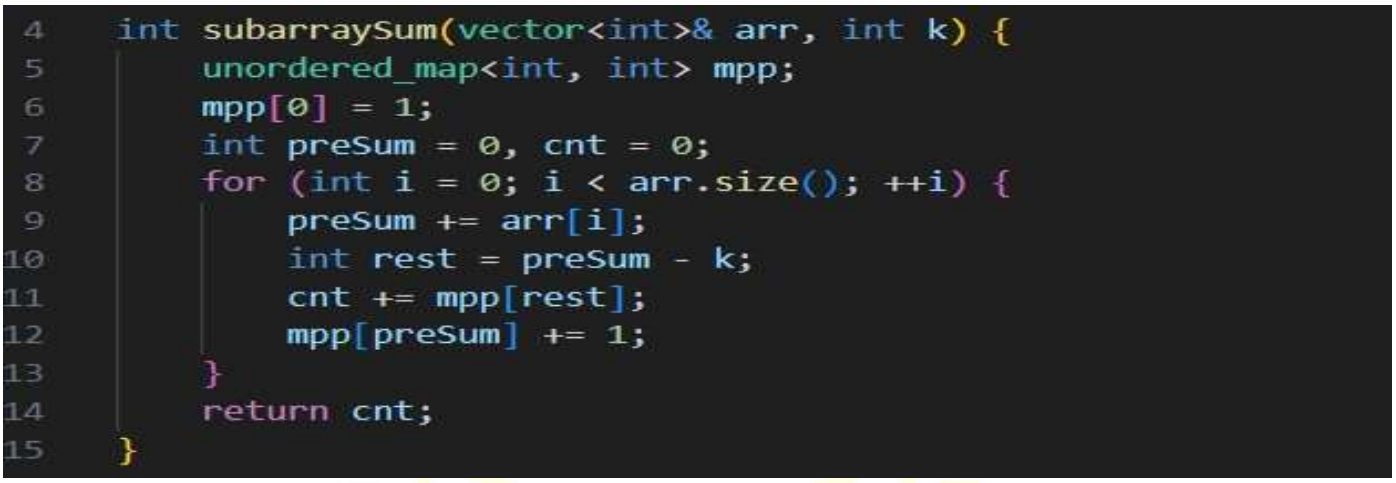
Input: nums =123-3111

K=3

Output: 6

Approach-

In this approach, the function calculates the number of subarrays whose sum equals a given value `k`. It uses a hash map to keep track of prefix sums and their frequencies. As the function iterates through the array, it maintains a running total (`preSum`) of the elements encountered. For each element, it checks if there is a previous prefix sum that would make the current subarray sum equal to `k` by subtracting `k` from `preSum` and looking it up in the hash map. The frequency of such prefix sums is added to the count. Finally, the hash map is updated with the current prefix sum. This method efficiently computes the count in linear time.



```
int subarraySum(vector<int>& arr, int k) {
    unordered_map<int, int> mpp;
    mpp[0] = 1;
    int preSum = 0, cnt = 0;
    for (int i = 0; i < arr.size(); ++i) {
        preSum += arr[i];
        int rest = preSum - k;
        cnt += mpp[rest];
        mpp[preSum] += 1;
    }
    return cnt;
}
```