Q-1: You are tasked to write a Program to find the sum of elements above and below the main diagonal of a matrix.

Sample test case:

|  |
| --- |
| Input: {{1, 2, 3},  {4, 5, 6},  {7, 8, 9}}  Output:  Sum above diagonal: 11  Sum below diagonal: 19 |

Solution:

#include <iostream>

// Function to find the sum of elements above and below the main diagonal of a matrix using pointer arithmetic

void findDiagonalSums(int\* matrix, int size) {

int sumAboveDiagonal = 0;

int sumBelowDiagonal = 0;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (j > i) {

sumAboveDiagonal += \*(matrix + i \* size + j);

} else if (j < i) {

sumBelowDiagonal += \*(matrix + i \* size + j);

}

}

}

std::cout << "Sum above diagonal: " << sumAboveDiagonal << std::endl;

std::cout << "Sum below diagonal: " << sumBelowDiagonal << std::endl;

}

int main() {

int matrix[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int size = sizeof(matrix[0]) / sizeof(matrix[0][0]);

findDiagonalSums(reinterpret\_cast<int\*>(matrix), size);

return 0;

}

Q-2: Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

Sample test case:

|  |
| --- |
| Input: nums = [2,7,11,15], target = 9  Output: [0,1] |

Solution:

#include <iostream>

using namespace std;

// Function to find the indices of two elements in the array that sum up to the target.

void targetSum(int a[], int target, int n, int res[])

{

for (int i = 0; i < n - 1; i++)

{

for (int j = i + 1; j < n; j++)

{

// Check if the sum of elements at indices i and j is equal to the target.

if (a[i] + a[j] == target)

{

// Store the indices of the elements in the 'res' array.

res[0] = i;

res[1] = j;

return; // Return immediately after finding the indices.

}

}

}

}

int main()

{

int a[] = {2, 7, 11, 15};

int n = sizeof(a) / sizeof(a[0]); // Calculate the size of the array 'a'.

int target = 9;

int res[2] = {}; // Create an array to store the indices of the elements that sum up to the target.

targetSum(a, target, n, res); // Call the function to find the indices.

// Print the indices of the elements that sum up to the target.

for (int i = 0; i < 2; i++)

{

cout << res[i] << " ";

}

cout << '\n';

}

Q-3: In a high school mathematics class, the teacher is introducing the concept of matrix transposition to the students.

Students are tasked to find transpose of matrix. Write a Program to help students.

Sample test case:

|  |
| --- |
| Input: matrix= {{1, 2},{3, 4},{5, 6}}  Output:  Transpose of the matrix:  1 3 5  2 4 6 |
| Solution:  #include <iostream>  using namespace std;  const int MAX\_SIZE = 100;  // Function to find the transpose of a matrix  void findTranspose(int matrix[][MAX\_SIZE], int rows, int cols) {  int transpose[MAX\_SIZE][MAX\_SIZE];  // Calculating the transpose  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  transpose[j][i] = matrix[i][j];  }  }  // Displaying the transpose  cout << "Transpose of the matrix:" << endl;  for (int i = 0; i < cols; i++) {  for (int j = 0; j < rows; j++) {  cout << transpose[i][j] << " ";  }  cout << endl;  }  }  int main() {  int matrix[MAX\_SIZE][MAX\_SIZE];  int rows, cols;  cout << "Enter the number of rows in the matrix: ";  cin >> rows;  cout << "Enter the number of columns in the matrix: ";  cin >> cols;  cout << "Enter the elements of the matrix:" << endl;  for (int i = 0; i < rows; i++) {  for (int j = 0; j < cols; j++) {  cin >> matrix[i][j];  }  }  findTranspose(matrix, rows, cols);  return 0;  } |

Q-4: Students are arranged in matrix form for assembly.

Help coach to find the maximum height student in each row of a students.

Sample test case:

|  |
| --- |
| Input: {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}  Output:  Maximum element in row 1: 3  Maximum element in row 2: 6  Maximum element in row 3: 9 |

Solution:

#include <iostream>

// Function to find the maximum element in each row of a matrix using pointer arithmetic

void findMaxInRows(int\* matrix, int rows, int cols) {

for (int i = 0; i < rows; i++) {

int maxElement = \*matrix;

for (int j = 1; j < cols; j++) {

if (\*(matrix + i \* cols + j) > maxElement) {

maxElement = \*(matrix + i \* cols + j);

}

}

std::cout << "Maximum element in row " << i + 1 << ": " << maxElement << std::endl;

}

}

int main() {

int matrix[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int rows = sizeof(matrix) / sizeof(matrix[0]);

int cols = sizeof(matrix[0]) / sizeof(matrix[0][0]);

findMaxInRows(reinterpret\_cast<int\*>(matrix), rows, cols);

return 0;

}

Q-5: In a school's annual sports event, the coach wants to determine the second-highest score achieved by the students in the long jump competition.

The coach asks each participant to perform a long jump and records their distances in meters.

The distances are stored in an array, and the coach needs a program to find the second largest distance among all the recorded jumps.

Sample test case:

|  |
| --- |
| Input: {5, 9, 3, 7, 1}  Output: Second Largest element: 7 |

Solution:

#include <iostream>

// Function to find the second largest element in an array using pointer arithmetic

int findSecondLargestElement(int\* arr, int size) {

int largest = \*arr; // Initialize the largest element with the first element

int secondLargest = -1; // Initialize the second largest element as -1

// Find the largest element

for (int i = 1; i < size; i++) {

if (\*(arr + i) > largest) {

secondLargest = largest;

largest = \*(arr + i);

} else if (\*(arr + i) < largest && \*(arr + i) > secondLargest) {

secondLargest = \*(arr + i);

}

}

return secondLargest;

}

int main() {

int arr[] = {5, 9, 3, 7, 1};

int size = sizeof(arr) / sizeof(arr[0]);

int secondLargest = findSecondLargestElement(arr, size);

if (secondLargest != -1) {

std::cout << "Second Largest element: " << secondLargest << std::endl;

} else {

std::cout << "No second largest element exists." << std::endl;

}

return 0;

}

Q-6: Given an array of integers containing only 0s and 1s,write a program to segregate the 0s and 1s in the array, where all the 0s should come before all the 1s.

Sample test case:

|  |
| --- |
| Input: {1, 0, 1, 0, 1, 0, 0, 1}  Output: Segregated Array: 0 0 0 0 1 1 1 1 |

Constraints:

* Input array should contain only 0 and 1.

Solution:

#include <iostream>

// Function to segregate 0s and 1s in an array

void segregateZerosAndOnes(int arr[], int size) {

int countZeros = 0;

// Count the number of zeros

for (int i = 0; i < size; i++) {

if (arr[i] == 0) {

countZeros++;

}

}

// Place the zeros at the beginning of the array

for (int i = 0; i < countZeros; i++) {

arr[i] = 0;

}

// Place the ones after the zeros

for (int i = countZeros; i < size; i++) {

arr[i] = 1;

}

}

int main() {

int arr[] = {1, 0, 1, 0, 1, 0, 0, 1};

int size = sizeof(arr) / sizeof(arr[0]);

segregateZerosAndOnes(arr, size);

std::cout << "Segregated Array: ";

for (int i = 0; i < size; i++) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

return 0;

}

Q-7: Given an array of integers, write a program to find the maximum product of two integers in the array.

Sample test case:

|  |
| --- |
| Input: {1, 2, 3, 4, 5}  Output: Maximum Product: 20 |

Solution:

#include <iostream>

// Function to find the maximum product of two integers in an array

int findMaxProduct(int arr[], int size) {

if (size < 2) {

return -1; // Not enough elements in the array

}

int maxProduct = arr[0] \* arr[1];

for (int i = 0; i < size; i++) {

for (int j = i + 1; j < size; j++) {

if (arr[i] \* arr[j] > maxProduct) {

maxProduct = arr[i] \* arr[j];

}

}

}

return maxProduct;

}

int main() {

int arr[] = {1, 2, 3, 4, 5};

int size = sizeof(arr) / sizeof(arr[0]);

int maxProduct = findMaxProduct(arr, size);

std::cout << "Maximum Product: " << maxProduct << std::endl;

return 0;

}

Q-8: Given an array of integers, write a program to find the length of the longest increasing subarray in the array.

Sample test case:

|  |
| --- |
| Input: {5, 6, 3, 5, 7, 8, 9, 1, 2}  Output: Length of Longest Increasing Subarray: 5 |

Solution:

#include <iostream>

// Function to find the length of the longest increasing subarray

int findLongestIncreasingSubarray(int arr[], int size) {

int maxLength = 1;

int currentLength = 1;

for (int i = 1; i < size; i++) {

if (arr[i] > arr[i - 1]) {

currentLength++;

} else {

maxLength = std::max(maxLength, currentLength);

currentLength = 1;

}

}

return std::max(maxLength, currentLength);

}

int main() {

int arr[] = {5, 6, 3, 5, 7, 8, 9, 1, 2};

int size = sizeof(arr) / sizeof(arr[0]);

int longestIncreasingSubarray = findLongestIncreasingSubarray(arr, size);

std::cout << "Length of Longest Increasing Subarray: " << longestIncreasingSubarray << std::endl;

return 0;

}

Q-9: Program to find the equilibrium index of an array,

where the equilibrium index is the position where the sum of elements to the left is equal to the sum of elements to the right.

Sample test case:

|  |
| --- |
| Input: {-7, 1, 5, 2, -4, 3, 0}  Output: Equilibrium Index: 3 |

Solution:

#include <iostream>

// Function to find the equilibrium index of an array

int findEquilibriumIndex(int arr[], int size) {

int totalSum = 0;

int leftSum = 0;

// Calculate the total sum of array elements

for (int i = 0; i < size; i++) {

totalSum += arr[i];

}

for (int i = 0; i < size; i++) {

totalSum -= arr[i]; // Reduce the total sum by the current element

if (leftSum == totalSum) {

return i; // Equilibrium index found

}

leftSum += arr[i]; // Add the current element to the left sum

}

return -1; // Equilibrium index not found

}

int main() {

int arr[] = {-7, 1, 5, 2, -4, 3, 0};

int size = sizeof(arr) / sizeof(arr[0]);

int equilibriumIndex = findEquilibriumIndex(arr, size);

std::cout << "Equilibrium Index: " << equilibriumIndex << std::endl;

return 0;

}

Q-10: Program to find the maximum difference between two elements in an array,

where the larger element appears after the smaller element:

Sample test case:

|  |
| --- |
| Input: {2, 3, 10, 6, 4, 8, 1}  Output: Maximum Difference: 8 |

Solution:

#include <iostream>

// Function to find the maximum difference between two elements in an array

int findMaxDifference(int arr[], int size) {

int minElement = arr[0];

int maxDiff = arr[1] - arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] - minElement > maxDiff) {

maxDiff = arr[i] - minElement;

}

if (arr[i] < minElement) {

minElement = arr[i];

}

}

return maxDiff;

}

int main() {

int arr[] = {2, 3, 10, 6, 4, 8, 1};

int size = sizeof(arr) / sizeof(arr[0]);

int maxDifference = findMaxDifference(arr, size);

std::cout << "Maximum Difference: " << maxDifference << std::endl;

return 0;

}

Q-11: In a school, the teacher wants to calculate the average test score of a group of students. The teacher has already collected the test scores of each student and stored them in an array. However, to save memory and improve performance, the teacher decides to use pointers to calculate the average instead of copying the entire array.

Give a program to fulfill teacher's requirements.

Sample test case:

|  |
| --- |
| Input: 1, 2, 3, 4, 5  Output: Average of array elements: 3 |

Solution:

#include <iostream>

// Function to find the average of an array

double findAverage(int\* arr, int size) {

int sum = 0; // Variable to store the sum of array elements

// Iterate through each element of the array

for (int i = 0; i < size; i++) {

sum += \*(arr + i); // Add the current element to the sum

}

return static\_cast<double>(sum) / size; // Calculate and return the average

}

int main() {

int arr[] = {1, 2, 3, 4, 5}; // Declare and initialize an integer array

int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array

double average = findAverage(arr, size); // Call the function to find the average

std::cout << "Average of array elements: " << average << std::endl; // Print the average

return 0; // Return 0 to indicate successful execution

}

Q-12: Emma, a student, has been assigned a task by her computer science teacher.

The task is to write a program that counts the number of words in a given string.

To make it more interesting, her teacher challenges her to use pointers in the program.

Help Emma to complete the task.

Sample test case:

|  |
| --- |
| Input: "Hello, how are you?"  Output: Number of words: 4 |

Solution:

#include <iostream>

#include <cstring>

#include <cctype>

// Function to count the number of words in a string

int countWords(char\* str) {

int count = 0; // Counter to keep track of the number of words

bool inWord = false; // Flag to track whether currently inside a word or not

// Iterate through each character in the string until the null terminator is reached

while (\*str) {

if (std::isalpha(\*str)) { // Check if the current character is alphabetic

if (!inWord) { // If not already in a word, increment the word count

count++;

inWord = true; // Set the flag to indicate being inside a word

}

} else {

inWord = false; // Reset the flag when encountering a non-alphabetic character

}

str++; // Move to the next character in the string

}

return count; // Return the total count of words

}

int main() {

char str[] = "Hello, how are you?"; // Create a character array (C-style string)

int count = countWords(str); // Call the function to count the words

std::cout << "Number of words: " << count << std::endl; // Print the word count

return 0;

}

Q-13: In a retail store, the manager wants to calculate the total sales made in a day.

The sales data is stored in an integer array. Instead of using the traditional indexing method, the store manager decides to challenge the sales team to use pointer arithmetic to calculate the total sales.

Sample test case:

|  |
| --- |
| Input: {1, 2, 3, 4, 5}  Output: Sum of array elements: 15 |

Solution:

#include <iostream>

int main() {

int arr[] = {1, 2, 3, 4, 5};

int\* ptr = arr;

int sum = 0;

while (ptr < arr + 5) {

sum += \*ptr;

ptr++;

}

std::cout << "Sum of array elements: " << sum << std::endl;

return 0;

}

Q-14: Given two integers A and B. The task is to swap two numbers. Swapping here means to interchange the values of A and B.

Sample test case:

|  |
| --- |
| Before swap - x: 10, y: 20  After swap - x: 20, y: 10 |

Solution:

#include <iostream>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 10;

int y = 20;

std::cout << "Before swap - x: " << x << ", y: " << y << std::endl;

swap(&x, &y);

std::cout << "After swap - x: " << x << ", y: " << y << std::endl;

return 0;

}

Q-15: Write a CPP function to Increment each element of an integer array by 1 by using pointer arithmetic operation.

Sample test case:

|  |
| --- |
| Input: {1, 2, 3, 4, 5}  Output: Modified array: 2 3 4 5 6 |

Solution:

#include <iostream>

int main() {

int arr[] = {1, 2, 3, 4, 5};

int\* ptr = arr;

for (int i = 0; i < 5; i++) {

\*ptr += 1;

ptr++;

}

std::cout << "Modified array: ";

for (int i = 0; i < 5; i++) {

std::cout << arr[i] << " ";

}

return 0;

}

Q-16: Write a program that takes an array, its size, and the number of rotations as input, and rotates the array left by the given number of positions using pointer arithmetic. Output the rotated array.

Sample test case:

|  |
| --- |
| Input:  Enter the size of the array: 6  Enter the elements of the array: 2 3 6 4 1 2  Enter the number of rotations: 2  Output: Rotated array: 6 4 1 2 2 3 |

Solution:

#include <iostream>

void rotateArray(int\* arr, int size, int rotations) {

rotations %= size;

if (rotations < 0) {

rotations += size;

}

int\* temp = new int[rotations];

for (int i = 0; i < rotations; ++i) {

temp[i] = arr[i];

}

for (int i = rotations; i < size; ++i) {

arr[i - rotations] = arr[i];

}

for (int i = 0; i < rotations; ++i) {

arr[size - rotations + i] = temp[i];

}

delete[] temp;

}

int main() {

int size, rotations;

std::cout << "Enter the size of the array: ";

std::cin >> size;

int\* arr = new int[size];

std::cout << "Enter the elements of the array: ";

for (int i = 0; i < size; ++i) {

std::cin >> arr[i];

}

std::cout << "Enter the number of rotations: ";

std::cin >> rotations;

// Rotate the array using pointer arithmetic

rotateArray(arr, size, rotations);

std::cout << "Rotated array: ";

for (int i = 0; i < size; ++i) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

delete[] arr;

return 0;

}

Q-17: In a language learning application, a new word game called "Palindromic Word Detector" is introduced to challenge users' language and programming skills. The game requires users to enter a word, and the application will check if the word is a palindrome using pointers.

Sample test case:

|  |
| --- |
| Input: Madam  Output: Palindrome |

Solution:

#include <iostream>

#include <cstring>

#include <cctype>

// Function to check if a string is a palindrome

bool isPalindrome(char\* str) {

int length = strlen(str); // Calculate the length of the string

char\* start = str; // Pointer to the start of the string

char\* end = str + length - 1; // Pointer to the end of the string

// Compare characters from the start and end pointers until they meet in the middle

while (start < end) {

if (std::tolower(\*start) != std::tolower(\*end)) { // Compare characters while ignoring case

return false; // If a mismatch is found, the string is not a palindrome

}

start++; // Move the start pointer to the next character

end--; // Move the end pointer to the previous character

}

return true; // If the loop completes without finding any mismatches, the string is a palindrome

}

int main() {

char str[] = "Madam"; // Create a character array (C-style string)

bool palindrome = isPalindrome(str); // Call the function to check if the string is a palindrome

if (palindrome) {

std::cout << "Palindrome" << std::endl; // Print a message indicating that the string is a palindrome

} else {

std::cout << "Not a palindrome" << std::endl; // Print a message indicating that the string is not a palindrome

}

return 0;

}

Q-18: We have two sorted arrays, it needs to be merged into a single sorted array.

Write a program to merge two sorted arrays using dynamic memory allocation and pointers.

Sample test case:

|  |
| --- |
| Input: arr1= 2 4 6, arr2=3  Output:  Merged Array: 1 2 3 4 6 |

Constraints:

* Input array should be in sorted form only.

Solution:

#include <iostream>

// Function to merge two sorted arrays

int\* mergeArrays(const int\* arr1, int size1, const int\* arr2, int size2) {

// Calculate the size of the merged array

int mergedSize = size1 + size2;

// Allocate memory for the merged array dynamically

int\* mergedArray = new int[mergedSize];

// Pointers to traverse the input arrays

const int\* ptr1 = arr1;

const int\* ptr2 = arr2;

// Pointer to traverse the merged array

int\* mergedPtr = mergedArray;

// Merge the two sorted arrays into the merged array

while (ptr1 < arr1 + size1 && ptr2 < arr2 + size2) {

if (\*ptr1 <= \*ptr2) {

\*mergedPtr = \*ptr1;

ptr1++;

} else {

\*mergedPtr = \*ptr2;

ptr2++;

}

mergedPtr++;

}

// Copy the remaining elements of the first array, if any

while (ptr1 < arr1 + size1) {

\*mergedPtr = \*ptr1;

ptr1++;

mergedPtr++;

}

// Copy the remaining elements of the second array, if any

while (ptr2 < arr2 + size2) {

\*mergedPtr = \*ptr2;

ptr2++;

mergedPtr++;

}

return mergedArray;

}

int main() {

int size1, size2;

std::cout << "Enter the size of the first sorted array: ";

std::cin >> size1;

// Allocate memory for the first array dynamically

int\* arr1 = new int[size1];

std::cout << "Enter the elements of the first sorted array:" << std::endl;

for (int i = 0; i < size1; i++) {

std::cin >> arr1[i];

}

std::cout << "Enter the size of the second sorted array: ";

std::cin >> size2;

// Allocate memory for the second array dynamically

int\* arr2 = new int[size2];

std::cout << "Enter the elements of the second sorted array:" << std::endl;

for (int i = 0; i < size2; i++) {

std::cin >> arr2[i];

}

// Call the mergeArrays function to get the merged sorted array

int\* mergedArray = mergeArrays(arr1, size1, arr2, size2);

std::cout << "Merged sorted array:" << std::endl;

for (int i = 0; i < size1 + size2; i++) {

std::cout << mergedArray[i] << " ";

}

// Deallocate the dynamically allocated memory

delete[] arr1;

delete[] arr2;

delete[] mergedArray;

return 0;

}