**SALEM MAINA**

**SCT212-0167/2022**

**DATA STRUCTURES AND ALGORITHMS LAB 1 ASSIGNMENT**

1.

#include <iostream>

#include <limits>

int summation(int arr[], int n) {

int sum = 0;

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

sum += arr[i];

}

return sum;

}

int maximum(int arr[], int n) {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

int main() {

int n;

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

std::cin >> n;

if (n <= 0 || n > std::numeric\_limits<int>::max()) {

std::cout << "Invalid array length\n";

return 1;

}

int arr[n];

std::cout << "Enter " << n << " integers:\n";

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

std::cin >> arr[i];

}

std::cout << "Sum of the integers: " << summation(arr, n) << std::endl;

std::cout << "Maximum integer: " << maximum(arr, n) << std::endl;

return 0;

}

2.

#include <iostream>

#include <vector>

void reverse(std::vector<int>& nums, int start, int end) {

while (start < end) {

std::swap(nums[start], nums[end]);

start++;

end--;

}

}

void rotate(std::vector<int>& nums, int k) {

int n = nums.size();

k = k % n; // Calculate the actual number of rotations needed

// Reverse the entire array

reverse(nums, 0, n - 1);

// Reverse the first k elements

reverse(nums, 0, k - 1);

// Reverse the remaining n - k elements

reverse(nums, k, n - 1);

}

int main() {

std::vector<int> nums = {1, 2, 3, 4, 5};

int k = 3;

rotate(nums, k);

std::cout << "Rotated Array: ";

for (int num : nums) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

3.

#include <stdio.h>

#include <stdbool.h>

bool containsDuplicate(int\* nums, int numsSize) {

// Initialize an array to keep track of visited numbers

bool visited[100000] = {false}; // Assuming the range of integers is within [-100000, 100000]

// Iterate through the array

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

// If the current number has been visited before, return true

if (visited[nums[i]]) {

return true;

}

// Mark the current number as visited

visited[nums[i]] = true;

}

// If no duplicates are found after iterating through the array, return false

return false;

}

int main() {

int nums1[] = {1, 2, 3, 4, 5};

int nums2[] = {1, 2, 3, 4, 1};

int numsSize1 = sizeof(nums1) / sizeof(nums1[0]);

int numsSize2 = sizeof(nums2) / sizeof(nums2[0]);

// Test the containsDuplicate function

printf("%s\n", containsDuplicate(nums1, numsSize1) ? "true" : "false"); // Output: false

printf("%s\n", containsDuplicate(nums2, numsSize2) ? "true" : "false"); // Output: true

return 0;

}

4.

#include <stdio.h>

int singleNumber(int\* nums, int numsSize) {

int result = 0;

// XOR all the numbers in the array

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

result ^= nums[i];

}

return result;

}

int main() {

int nums[] = {4, 1, 2, 1, 2};

int numsSize = sizeof(nums) / sizeof(nums[0]);

// Find the single number

int single = singleNumber(nums, numsSize);

// Print the result

printf("The single number is: %d\n", single); // Output: 4

return 0;

}