1. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Function to compute the difference array for a given word

void computeDifferenceArray(char\* word, int\* diffArray, int length) {

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

diffArray[i] = word[i + 1] - word[i];

}

}

// Function to compare two difference arrays

int areArraysEqual(int\* arr1, int\* arr2, int length) {

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

if (arr1[i] != arr2[i]) {

return 0;

}

}

return 1;

}

// Function to find the odd string out

char\* findOddString(char\*\* words, int wordsSize) {

int length = strlen(words[0]);

int diffArrays[wordsSize][length - 1];

// Compute difference arrays for all words

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

computeDifferenceArray(words[i], diffArrays[i], length);

}

// Check the difference arrays to find the odd one

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

int isOdd = 1;

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

if (i != j && areArraysEqual(diffArrays[i], diffArrays[j], length - 1)) {

isOdd = 0;

break;

}

}

if (isOdd) {

return words[i];

}

}

return NULL;

}

int main() {

// Example 1

char\* words1[] = {"adc", "wzy", "abc"};

int wordsSize1 = sizeof(words1) / sizeof(words1[0]);

printf("Output: %s\n", findOddString(words1, wordsSize1));

// Example 2

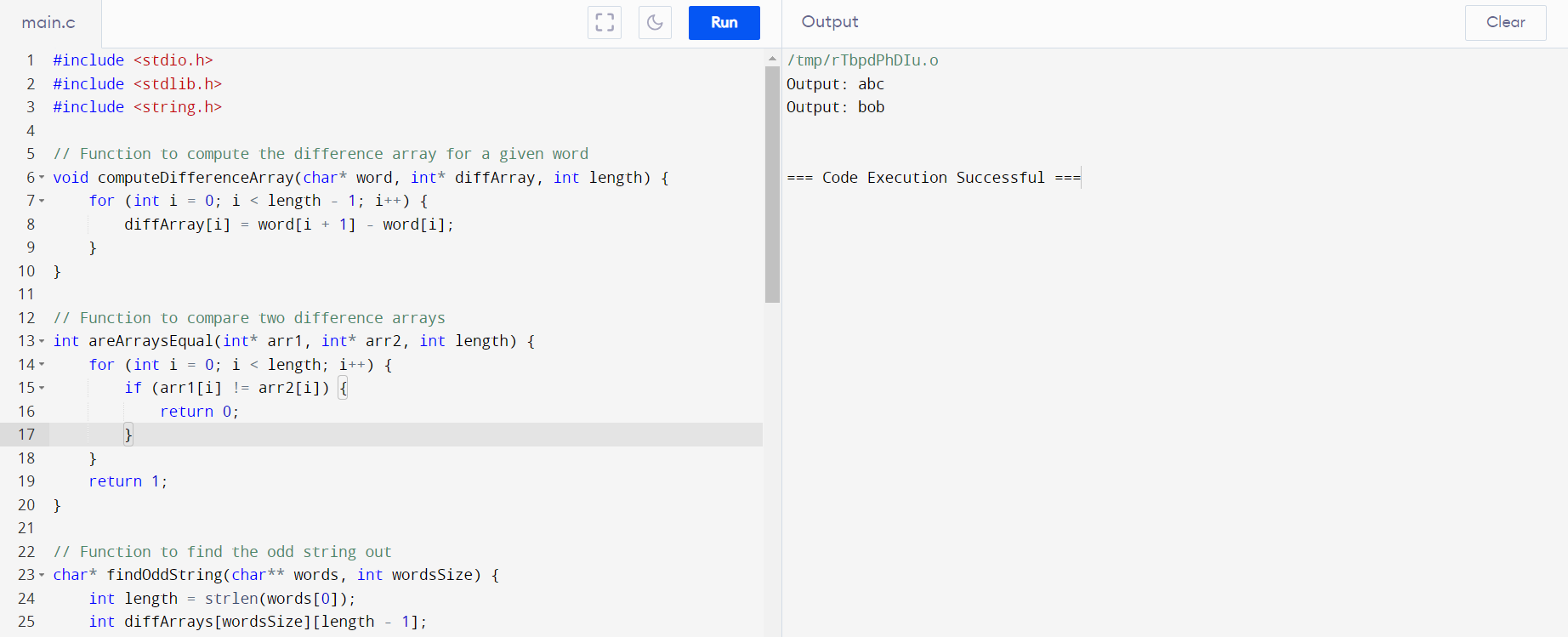
char\* words2[] = {"aaa", "bob", "ccc", "ddd"};

int wordsSize2 = sizeof(words2) / sizeof(words2[0]);

printf("Output: %s\n", findOddString(words2, wordsSize2));

return 0;

}



2. #include <stdio.h>

#include <string.h>

#include <stdlib.h>

int isWithinTwoEdits(char\* query, char\* dictWord, int length) {

int diffCount = 0;

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

if (query[i] != dictWord[i]) {

diffCount++;

if (diffCount > 2) return 0;

}

}

return 1;

}

char\*\* findWordsWithinTwoEdits(char\*\* queries, int queriesSize, char\*\* dictionary, int dictSize, int\* returnSize) {

int wordLength = strlen(queries[0]);

char\*\* result = (char\*\*)malloc(queriesSize \* sizeof(char\*));

\*returnSize = 0;

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

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

if (isWithinTwoEdits(queries[i], dictionary[j], wordLength)) {

result[(\*returnSize)++] = queries[i];

break;

}

}

}

return result;

}

int main() {

char\* queries[] = {"word", "note", "ants", "wood"};

char\* dictionary[] = {"wood", "joke", "moat"};

int queriesSize = sizeof(queries) / sizeof(queries[0]);

int dictSize = sizeof(dictionary) / sizeof(dictionary[0]);

int returnSize;

char\*\* result = findWordsWithinTwoEdits(queries, queriesSize, dictionary, dictSize, &returnSize);

printf("Output: [");

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

printf("\"%s\"", result[i]);

if (i < returnSize - 1) printf(", ");

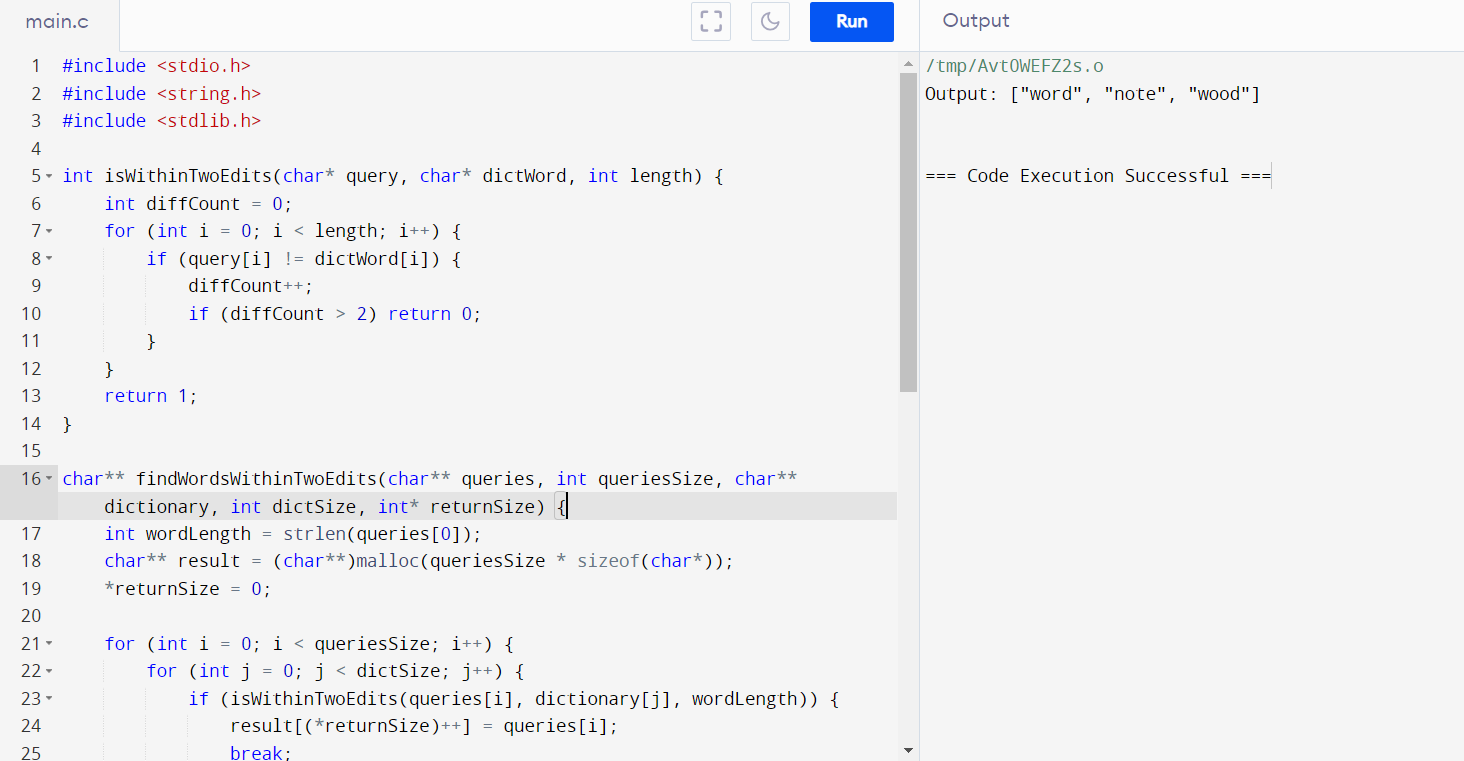
}

printf("]\n");

free(result);

return 0;

}



3. #include <stdio.h>

#include <limits.h>

int maxTargetsDestroyed(int\* nums, int numsSize, int space) {

int maxTargets = 0;

int minSeed = INT\_MAX;

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

int seed = nums[i];

int count = 0;

// Count targets that can be destroyed starting from nums[i]

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

if ((nums[j] - seed) % space == 0) {

count++;

}

}

// Update maximum targets destroyed and minimum seed

if (count > maxTargets || (count == maxTargets && seed < minSeed)) {

maxTargets = count;

minSeed = seed;

}

}

return minSeed;

}

int main() {

int nums1[] = {3, 7, 8, 1, 1, 5};

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

int space1 = 2;

printf("Output: %d\n", maxTargetsDestroyed(nums1, numsSize1, space1));

int nums2[] = {1, 3, 5, 2, 4, 6};

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

int space2 = 2;

printf("Output: %d\n", maxTargetsDestroyed(nums2, numsSize2, space2));

int nums3[] = {6, 2, 5};

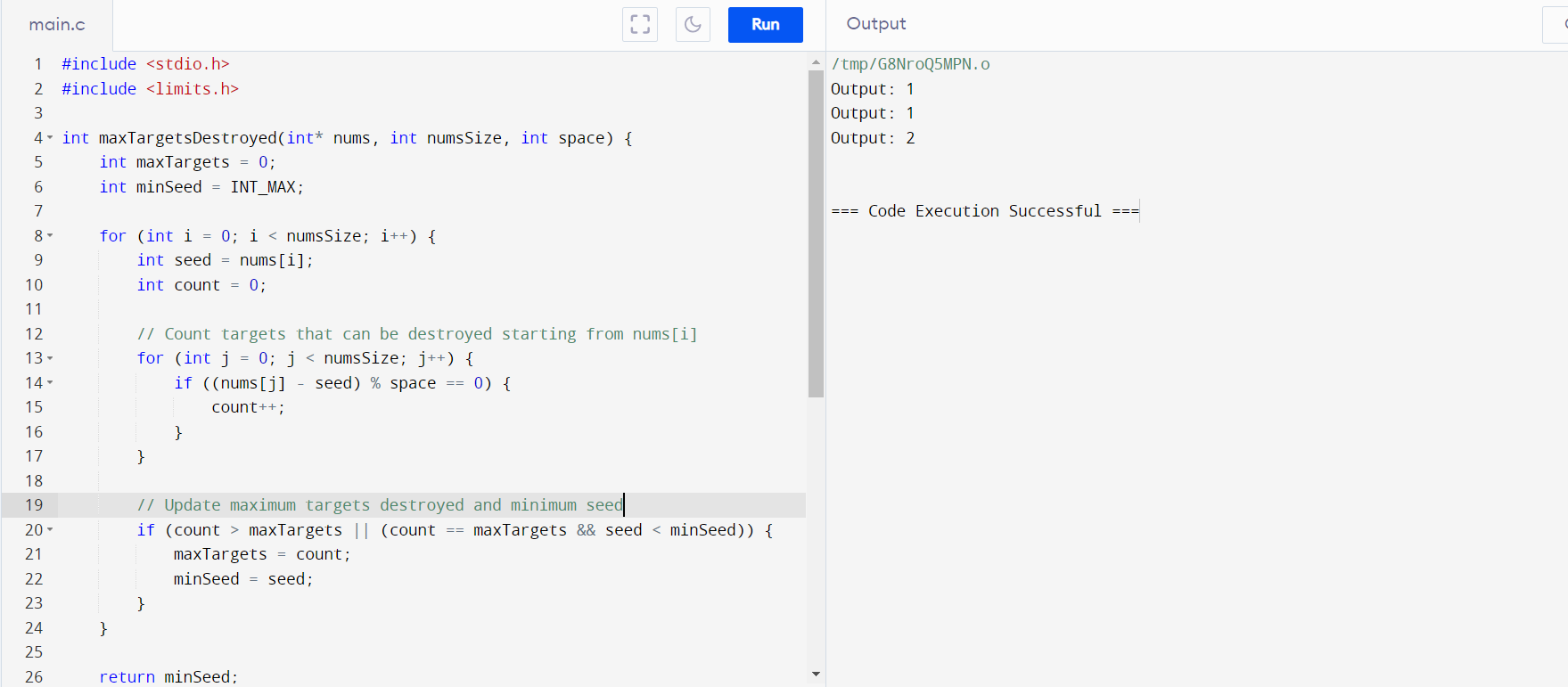
int numsSize3 = sizeof(nums3) / sizeof(nums3[0]);

int space3 = 100;

printf("Output: %d\n", maxTargetsDestroyed(nums3, numsSize3, space3));

return 0;

}



4. #include <stdio.h>

#include <stdlib.h>

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

int\* result = (int\*)malloc(numsSize \* sizeof(int));

int stack[numsSize]; // Stack to store indices

int top = -1; // Initialize stack top

// Initialize result array with -1

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

result[i] = -1;

}

// Traverse nums array to find next greater elements

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

// Pop elements from stack while they are smaller than nums[i]

while (top >= 0 && nums[stack[top]] < nums[i]) {

int idx = stack[top--];

if (top >= 0) {

result[stack[top]] = nums[i];

}

}

// Push current index onto stack

stack[++top] = i;

}

return result;

}

int main() {

// Example 1

int nums1[] = {2, 4, 0, 9, 6};

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

int\* result1 = nextGreaterElement(nums1, numsSize1);

printf("Output for Example 1: ");

for (int i = 0; i < numsSize1; i++) {

printf("%d ", result1[i]);

}

printf("\n");

free(result1);

// Example 2

int nums2[] = {3, 3};

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

int\* result2 = nextGreaterElement(nums2, numsSize2);

printf("Output for Example 2: ");

for (int i = 0; i < numsSize2; i++) {

printf("%d ", result2[i]);

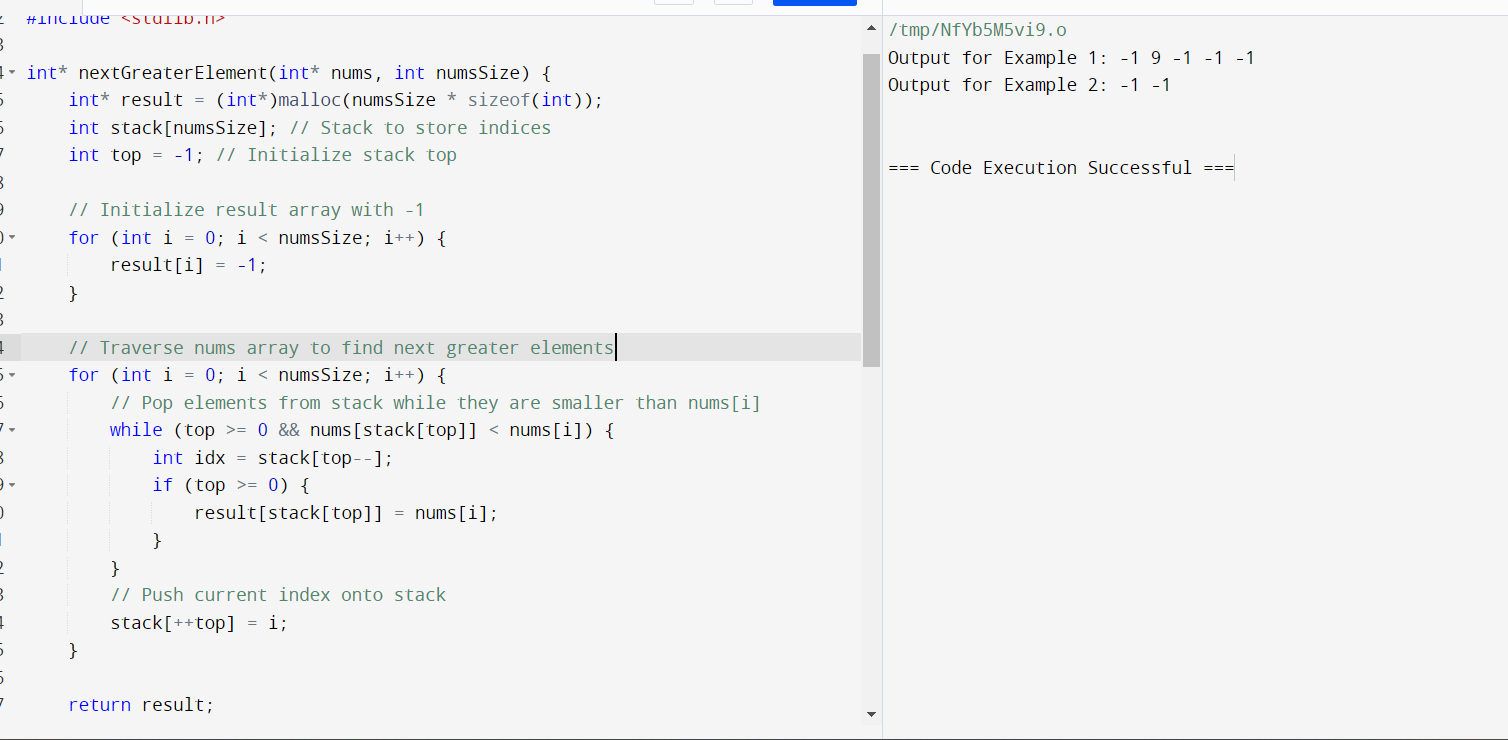
}

printf("\n");

free(result2);

return 0;

}



5. #include <stdio.h>

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

int sum = 0;

int count = 0;

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

if (nums[i] % 2 == 0 && nums[i] % 3 == 0) {

sum += nums[i];

count++;

}

}

if (count > 0) {

return sum / count;

} else {

return 0; // If no numbers satisfy the condition, return 0

}

}

int main() {

// Example 1

int nums1[] = {1, 3, 6, 10, 12, 15};

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

printf("Output for Example 1: %d\n", averageEvenDivisibleByThree(nums1, numsSize1));

// Example 2

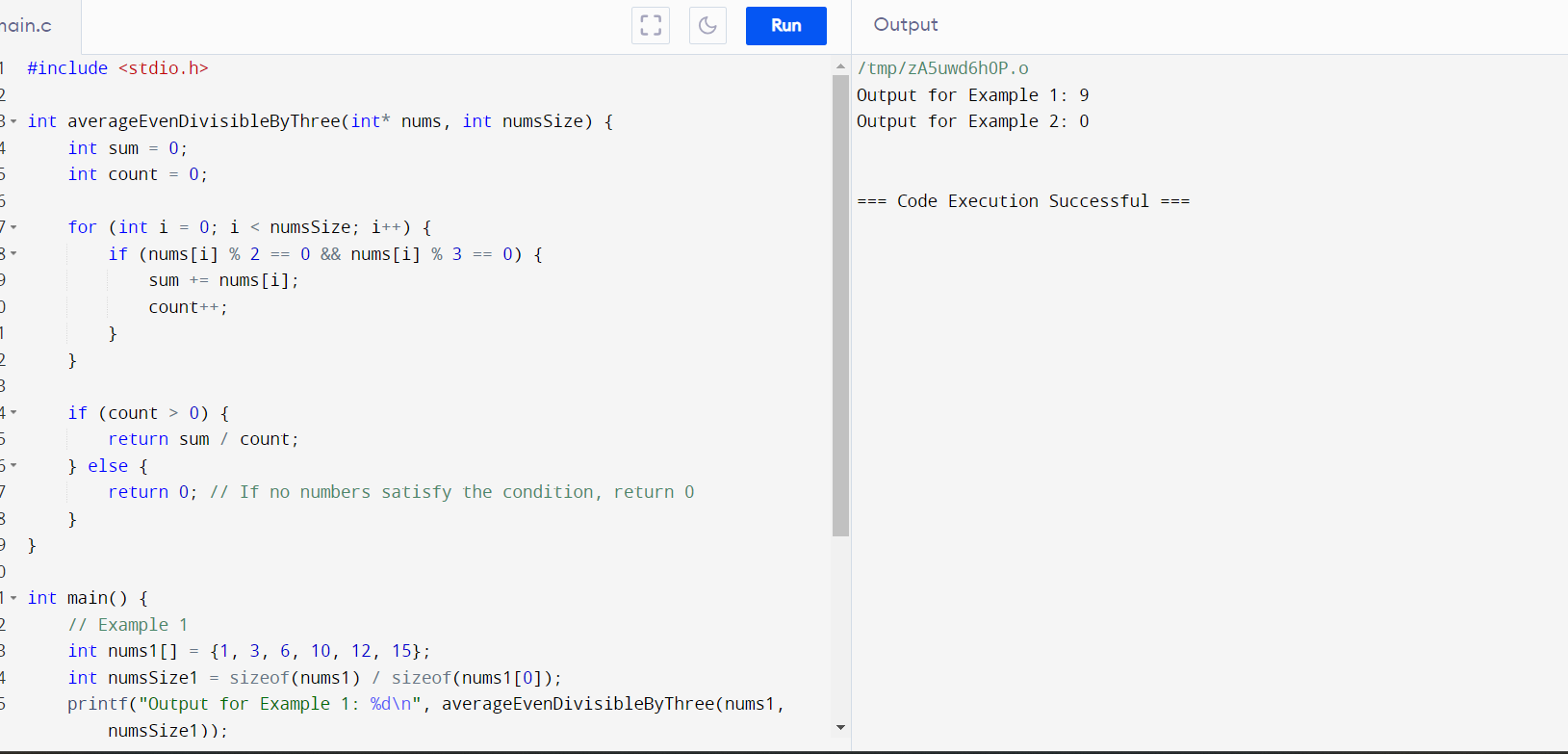
int nums2[] = {1, 2, 4, 7, 10};

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

printf("Output for Example 2: %d\n", averageEvenDivisibleByThree(nums2, numsSize2));

return 0;

}



6. #include <stdio.h>

#include <string.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_N 100000

#define MAX\_LEN 6 // Maximum length of creator and id strings (including null terminator)

// Structure to hold video information

typedef struct {

char creator[MAX\_LEN];

char id[MAX\_LEN];

int views;

} Video;

// Function to find the most popular video creator and their most viewed video id

char\*\* mostPopularVideoCreator(char\*\* creators, char\*\* ids, int\* views, int n, int\* returnSize) {

// Array to store the total views for each creator

int maxViews = 0;

int\* creatorViews = (int\*)calloc(n, sizeof(int));

// Populate creatorViews and find maxViews

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

creatorViews[i] += views[i];

if (creatorViews[i] > maxViews) {

maxViews = creatorViews[i];

}

}

// Find creators with maxViews

int count = 0;

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

if (creatorViews[i] == maxViews) {

count++;

}

}

// Allocate memory for the result array

char\*\* result = (char\*\*)malloc(count \* sizeof(char\*));

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

result[i] = (char\*)malloc(2 \* MAX\_LEN \* sizeof(char));

}

// Populate result array with creators and their most viewed video id

int index = 0;

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

if (creatorViews[i] == maxViews) {

strcpy(result[index], creators[i]);

strcat(result[index], " ");

strcat(result[index], ids[i]);

index++;

}

}

\*returnSize = count;

free(creatorViews);

return result;

}

int main() {

// Example 1

char\* creators1[] = {"alice", "bob", "alice", "chris"};

char\* ids1[] = {"one", "two", "three", "four"};

int views1[] = {5, 10, 5, 4};

int n1 = sizeof(creators1) / sizeof(creators1[0]);

int returnSize1;

char\*\* result1 = mostPopularVideoCreator(creators1, ids1, views1, n1, &returnSize1);

printf("Output for Example 1:\n");

for (int i = 0; i < returnSize1; i++) {

printf("[%s]\n", result1[i]);

}

printf("\n");

// Example 2

char\* creators2[] = {"alice", "alice", "alice"};

char\* ids2[] = {"a", "b", "c"};

int views2[] = {1, 2, 2};

int n2 = sizeof(creators2) / sizeof(creators2[0]);

int returnSize2;

char\*\* result2 = mostPopularVideoCreator(creators2, ids2, views2, n2, &returnSize2);

printf("Output for Example 2:\n");

for (int i = 0; i < returnSize2; i++) {

printf("[%s]\n", result2[i]);

}

printf("\n");

// Free memory allocated for results

for (int i = 0; i < returnSize1; i++) {

free(result1[i]);

}

free(result1);

for (int i = 0; i < returnSize2; i++) {

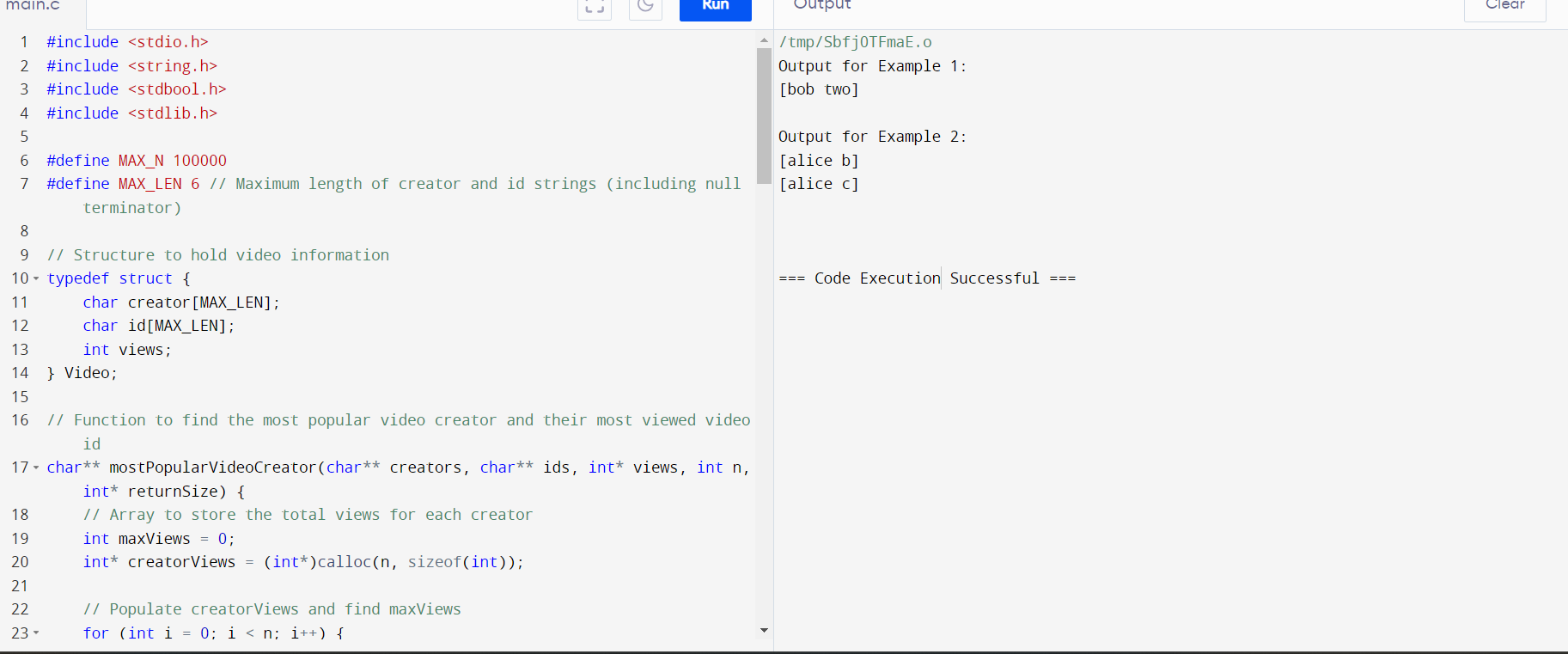
free(result2[i]);

}

free(result2);

return 0;

}



7. #include <stdio.h>

// Function to calculate the sum of digits of a number

int sumOfDigits(long long num) {

int sum = 0;

while (num > 0) {

sum += num % 10;

num /= 10;

}

return sum;

}

// Function to find the minimum non-negative integer x such that n + x is beautiful

int minimumAdditionToMakeBeautiful(long long n, int target) {

// Calculate the initial sum of digits of n

int sumDigits = sumOfDigits(n);

// Calculate the minimum x needed

int x = 0;

while (sumDigits > target) {

x++;

sumDigits = sumOfDigits(n + x);

}

return x;

}

int main() {

// Example 1

long n1 = 16;

int target1 = 6;

int result1 = minimumAdditionToMakeBeautiful(n1, target1);

printf("Output for Example 1: %d\n", result1);

// Example 2

long n2 = 467;

int target2 = 6;

int result2 = minimumAdditionToMakeBeautiful(n2, target2);

printf("Output for Example 2: %d\n", result2);

// Example 3

long n3 = 1;

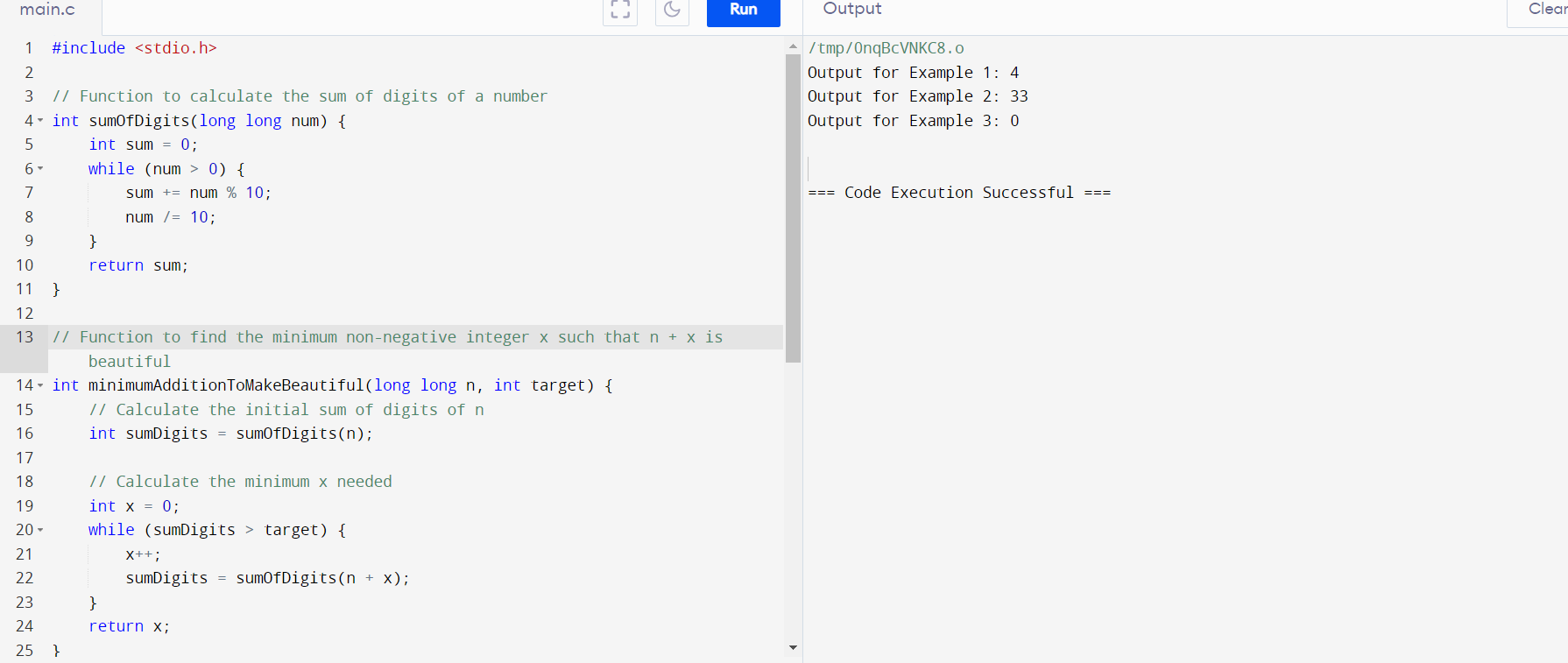
int target3 = 1;

int result3 = minimumAdditionToMakeBeautiful(n3, target3);

printf("Output for Example 3: %d\n", result3);

return 0;

}



8. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\*\* splitMessage(char\* message, int limit, int\* returnSize) {

int len = strlen(message);

int numParts = (len + limit - 1) / limit; // Calculate the number of parts needed

char\*\* result = (char\*\*)malloc(numParts \* sizeof(char\*));

int index = 0;

int i = 0;

while (i < len) {

int partLength = (i + limit <= len) ? limit : len - i; // Determine length of current part

result[index] = (char\*)malloc((partLength + 8) \* sizeof(char)); // +8 for "<a/b>"

// Create the current part

strncpy(result[index], message + i, partLength);

result[index][partLength] = '\0';

// Append suffix "<a/b>"

sprintf(result[index] + partLength, "<%d/%d>", index + 1, numParts);

i += partLength; // Move to the next segment of the message

index++;

}

\*returnSize = numParts;

return result;

}

int main() {

char message1[] = "this is really a very awesome message";

int limit1 = 9;

int returnSize1;

char\*\* result1 = splitMessage(message1, limit1, &returnSize1);

printf("Output for Example 1:\n");

for (int i = 0; i < returnSize1; i++) {

printf("%s\n", result1[i]);

free(result1[i]);

}

free(result1);

printf("\n");

char message2[] = "short message";

int limit2 = 15;

int returnSize2;

char\*\* result2 = splitMessage(message2, limit2, &returnSize2);

printf("Output for Example 2:\n");

for (int i = 0; i < returnSize2; i++) {

printf("%s\n", result2[i]);

free(result2[i]);

}

free(result2);

return 0;

}

