1. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_WORD\_LENGTH 21

#define MAX\_WORDS 101

// Function to compute the difference integer array for a given string

void computeDifferenceArray(char\* word, int\* diff) {

int len = strlen(word);

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

diff[i] = (word[i + 1] - 'a') - (word[i] - 'a');

}

}

int main() {

int n;

printf("Enter the number of words: ");

scanf("%d", &n);

char words[MAX\_WORDS][MAX\_WORD\_LENGTH];

printf("Enter the words:\n");

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

scanf("%s", words[i]);

}

int differences[MAX\_WORDS][MAX\_WORD\_LENGTH - 1];

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

computeDifferenceArray(words[i], differences[i]);

}

int oddIndex = -1;

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

int count = 0;

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

if (i != j && memcmp(differences[i], differences[j], sizeof(int) \* (strlen(words[0]) - 1)) != 0) {

count++;

}

}

if (count == n - 1) {

oddIndex = i;

break;

}

}

if (oddIndex != -1) {

printf("The odd string is: %s\n", words[oddIndex]);

} else {

printf("No odd string found.\n");

}

return 0;

}

2. #include <stdio.h>

#include <string.h>

#include <stdbool.h>

// Function to check if two words differ by at most two characters

bool isWithinTwoEdits(char\* word1, char\* word2) {

int count = 0;

for (int i = 0; word1[i] != '\0' && word2[i] != '\0'; i++) {

if (word1[i] != word2[i]) {

count++;

if (count > 2) {

return false;

}

}

}

return count <= 2;

}

void findWordsWithinTwoEdits(char\*\* queries, int queriesSize, char\*\* dictionary, int dictionarySize, char\*\* result, int\* resultSize) {

\*resultSize = 0;

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

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

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

result[\*resultSize] = queries[i];

(\*resultSize)++;

break;

}

}

}

}

int main() {

// Example input

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

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

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

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

// Array to hold the results

char\* result[queriesSize];

int resultSize = 0;

findWordsWithinTwoEdits(queries, queriesSize, dictionary, dictionarySize, result, &resultSize);

// Print the results

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

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

}

return 0;

}

3. #include <stdio.h>

#include <stdlib.h>

// Function to find the second greater element for each element in the array

void findSecondGreater(int\* nums, int numsSize, int\* result) {

// Allocate memory for two stacks

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

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

int firstTop = -1, secondTop = -1;

// Initialize the result array with -1

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

result[i] = -1;

}

// Traverse the array from right to left

for (int i = numsSize - 1; i >= 0; i--) {

// Move elements from secondStack to result if they are the second greater element

while (secondTop >= 0 && nums[secondStack[secondTop]] <= nums[i]) {

secondTop--;

}

if (secondTop >= 0) {

result[i] = nums[secondStack[secondTop]];

}

// Move elements from firstStack to secondStack

while (firstTop >= 0 && nums[firstStack[firstTop]] <= nums[i]) {

secondStack[++secondTop] = firstStack[firstTop--];

}

// Push current element index to firstStack

firstStack[++firstTop] = i;

}

// Free the allocated memory for stacks

free(firstStack);

free(secondStack);

}

int main() {

// Example input

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

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

int result[numsSize];

// Find the second greater elements

findSecondGreater(nums, numsSize, result);

// Print the result

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

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

}

printf("\n");

return 0;

}

4. #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 addition to make the number beautiful

long long makeBeautiful(int n, int target) {

long long addition = 0;

while (sumOfDigits(n + addition) > target) {

addition++;

}

return addition;

}

int main() {

// Example input

int n = 16;

int target = 6;

// Find the minimum addition

long long result = makeBeautiful(n, target);

// Print the result

printf("%lld\n", result);

return 0;

}

5. #include <stdio.h>

int minOperations(int\* nums, int n) {

int operations = 0;

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

// Check if current element is in the correct position

if (nums[i] == i) {

continue;

}

// If current element needs to be moved, find the correct position

int correctIndex = nums[i];

// Handle cases where the empty space is involved

if (correctIndex == 0) {

// Check if the empty space is at the end (nums[n - 1] == 0)

if (nums[n - 1] == 0) {

// Swap current element with the element at the end (already empty)

int temp = nums[i];

nums[i] = nums[n - 1];

nums[n - 1] = temp;

operations++;

} else {

// Find the element that should be in the empty space

int j = i + 1;

while (nums[j] != 0 && j < n) {

j++;

}

// Swap current element with the element that should be empty

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

operations++;

}

} else {

// Swap current element with the element at its correct position

int temp = nums[i];

nums[i] = nums[correctIndex];

nums[correctIndex] = temp;

operations++;

}

}

return operations;

}

int main() {

int nums[] = {4, 2, 0, 3, 1};

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

int minOps = minOperations(nums, n);

printf("Minimum operations needed: %d\n", minOps);

return 0;

}