**Add 2 numbers in array function**

int\* twoSum(int\* nums, int numsSize, int target, int\* returnSize)

{

for (int x = 0; x < numsSize - 1; x++)

{

for (int y = x + 1; y < numsSize; y++)

{

int sum = nums[x] + nums[y];

if (sum == target)

{

int\* result = (int\*)malloc(2 \* sizeof(int));

result[0] = x;

result[1] = y;

\*returnSize = 2;

return result;

}

}

}

returnSize = 0;

}

**Remove Duplicates from Sorted Array**

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

if(numsSize == 0)

{

return 0;

}

int k = 1;

for (int i= 1; i< numsSize; i++)

{

if(nums[i] != nums[i-1])

{

nums[k] = nums[i];

k++;

}

}

return k;

}

**Remove elements**

int removeElement(int\* nums, int numsSize, int val)

{

int newSize = 0;

for (int x = 0; x < numsSize; x++)

{

if (nums[x] != val)

{

nums[newSize] = nums[x];

newSize++;

}

}

return newSize;

}

**Roman to integer**

int romanToInt(char \* s){

int values[7] = {1000, 500, 100, 50, 10, 5, 1};

char symbols[7] = {'M', 'D', 'C', 'L', 'X', 'V', 'I'};

int result = 0;

int prevValue = 0;

int length = strlen(s);

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

int currentSymbolValue = 0;

for (int j = 0; j < 7; j++) {

if (symbols[j] == s[i]) {

currentSymbolValue = values[j];

break;

}

}

if (currentSymbolValue >= prevValue) {

result += currentSymbolValue;

} else {

result -= currentSymbolValue;

}

prevValue = currentSymbolValue;

}

return result;

}

**Longest common prefix**

char \* longestCommonPrefix(char \*\* strs, int strsSize){

if (strsSize == 0) {

// If the array is empty, return an empty string

return "";

}

int minLen = strlen(strs[0]);

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

// Find the minimum length among the strings

int len = strlen(strs[i]);

if (len < minLen) {

minLen = len;

}

}

char\* result = (char\*)malloc((minLen + 1) \* sizeof(char));

int index = 0;

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

char currentChar = strs[0][i];

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

if (strs[j][i] != currentChar) {

result[index] = '\0'; // Null-terminate the result

return result;

}

}

result[index++] = currentChar;

}

result[index] = '\0'; // Null-terminate the result

return result;

}

**Reverse integer**

#include <stdio.h>

#include <limits.h>

int reverse(int x) {

int reversed = 0;

while (x != 0) {

// Check for overflow/underflow before multiplying reversed by 10

if (reversed > INT\_MAX / 10 || reversed < INT\_MIN / 10) {

return 0;

}

int digit = x % 10;

x /= 10;

reversed = reversed \* 10 + digit;

}

return reversed;

}

int main() {

int x = 12345;

int reversed = reverse(x);

printf("Original: %d\n", x);

printf("Reversed: %d\n", reversed);

return 0;

}

**Count pairs whose sum is less than target**

int countPairs(int\* nums, int numsSize, int target){

int result = 0;

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

{

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

{

if (nums[i] + nums[j] < target)

{

result += 1;

}

}

}

return result;

}

**Account balance after rounded balance**

int accountBalanceAfterPurchase(int purchaseAmount){

int totalbalance = 100;

int balance = totalbalance - purchaseAmount ;

int remainder = 0;

if (balance >= 10)

{

remainder = balance % 10;

}

else

{

remainder = balance;

}

int result = 0;

if (remainder <= 5)

{

result = balance - remainder;

}

else

{

result = balance + (10 - remainder);

}

return result;

}

**No.of employ who met the target**

int numberOfEmployeesWhoMetTarget(int\* hours, int hoursSize, int target){

int result = 0;

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

{

if(hours[i] >= target)

{

result ++;

}

}

return result;

}