1. Implement a function that checks whether a given string is a palindrome or not.

use std::io;

fn is\_palindrome(s: &str) -> bool {

let s = s.to\_lowercase();

let reversed = s.chars().rev().collect::<String>();

s == reversed

}

fn main() {

println!("Enter a string:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let input = input.trim(); // Remove newline character

if is\_palindrome(input) {

println!("{} is a palindrome.", input);

} else {

println!("{} is not a palindrome.", input);

}

}

1. Given a sorted array of integers, implement a function that returns the index of the first occurrence of a given number.

use std::io;

fn first\_occurrence\_index(arr: &[i32], target: i32) -> Option<usize> {

for (index, &num) in arr.iter().enumerate() {

if num == target {

return Some(index);

} else if num > target {

break;

}

}

None

}

fn main() {

println!("Enter a sorted array of integers separated by spaces:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let sorted\_array: Vec<i32> = input

.split\_whitespace()

.filter\_map(|s| s.parse().ok())

.collect();

println!("Enter the target number:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let target\_number: i32 = input.trim().parse().expect("Invalid input");

if let Some(index) = first\_occurrence\_index(&sorted\_array, target\_number) {

println!("The first occurrence of {} is at index {}", target\_number, index);

} else {

println!("{} is not found in the array.", target\_number);

}

}

1. Given a string of words, implement a function that returns the shortest word in the string.

use std::io;

fn shortest\_word(string: &str) -> Option<&str> {

string.split\_whitespace().min\_by\_key(|word| word.len())

}

fn main() {

println!("Enter a string of words:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let input\_string = input.trim();

if let Some(shortest) = shortest\_word(&input\_string) {

println!("The shortest word is: {}", shortest);

} else {

println!("No words found in the string.");

}

}

1. Implement a function that checks whether a given number is prime or not.

use std::io;

fn is\_prime(num: u64) -> bool {

if num <= 1 {

return false;

}

for i in 2..=(num as f64).sqrt() as u64 {

if num % i == 0 {

return false;

}

}

true

}

fn main() {

println!("Enter a number:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let num: u64 = input.trim().parse().expect("Invalid input");

if is\_prime(num) {

println!("{} is a prime number.", num);

} else {

println!("{} is not a prime number.", num);

}

}

1. Given a sorted array of integers, implement a function that returns the median of the array.

use std::io;

fn median(arr: &[i32]) -> f64 {

let len = arr.len();

if len % 2 == 0 {

let mid = len / 2;

(arr[mid - 1] + arr[mid]) as f64 / 2.0

} else {

arr[len / 2] as f64

}

}

fn main() {

println!("Enter a sorted array of integers separated by spaces:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let sorted\_array: Vec<i32> = input

.split\_whitespace()

.filter\_map(|s| s.parse().ok())

.collect();

let median\_value = median(&sorted\_array);

println!("The median of the array is: {}", median\_value);

}

1. Implement a function that finds the longest common prefix of a given set of strings.

use std::io;

fn longest\_common\_prefix(strings: &[String]) -> String {

if strings.is\_empty() {

return String::new();

}

let first\_string = strings[0].as\_str();

let mut prefix = String::new();

'outer: for (i, char) in first\_string.chars().enumerate() {

for string in &strings[1..] {

if let Some(c) = string.chars().nth(i) {

if c != char {

break 'outer;

}

} else {

break 'outer;

}

}

prefix.push(char);

}

prefix

}

fn main() {

println!("Enter a set of strings separated by spaces:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let strings: Vec<String> = input

.split\_whitespace()

.map(|s| s.to\_string())

.collect();

let common\_prefix = longest\_common\_prefix(&strings);

if common\_prefix.is\_empty() {

println!("No common prefix found.");

} else {

println!("The longest common prefix is: {}", common\_prefix);

}

}

1. Implement a function that returns the kth smallest element in a given array.

use std::io;

fn kth\_smallest(arr: &[i32], k: usize) -> Option<i32> {

if k > arr.len() {

return None;

}

let mut sorted\_arr = arr.to\_vec();

sorted\_arr.sort();

Some(sorted\_arr[k - 1])

}

fn main() {

println!("Enter the array of integers separated by spaces:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let arr: Vec<i32> = input

.split\_whitespace()

.filter\_map(|s| s.parse().ok())

.collect();

println!("Enter the value of k:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let k: usize = input.trim().parse().expect("Invalid input");

if let Some(kth) = kth\_smallest(&arr, k) {

println!("The {}th smallest element in the array is: {}", k, kth);

} else {

println!("Invalid input: k is greater than the array length.");

}

}

1. Given a binary tree, implement a function that returns the maximum depth of the tree.

use std::io;

// Definition for a binary tree node.

#[derive(Debug)]

struct TreeNode {

val: i32,

left: Option<Box<TreeNode>>,

right: Option<Box<TreeNode>>,

}

impl TreeNode {

fn new(val: i32) -> Self {

TreeNode { val, left: None, right: None }

}

}

fn max\_depth(root: Option<&Box<TreeNode>>) -> i32 {

match root {

Some(node) => {

let left\_depth = max\_depth(node.left.as\_ref());

let right\_depth = max\_depth(node.right.as\_ref());

1 + left\_depth.max(right\_depth)

}

None => 0,

}

}

fn main() {

// Example binary tree: 3

// / \

// 9 20

// / \

// 15 7

let root = Some(Box::new(TreeNode {

val: 3,

left: Some(Box::new(TreeNode::new(9))),

right: Some(Box::new(TreeNode {

val: 20,

left: Some(Box::new(TreeNode::new(15))),

right: Some(Box::new(TreeNode::new(7))),

})),

}));

let depth = max\_depth(root.as\_ref());

println!("The maximum depth of the binary tree is: {}", depth);

}

1. Reverse a string in Rust

use std::io;

fn reverse\_string(input: &str) -> String {

input.chars().rev().collect()

}

fn main() {

println!("Enter a string:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let input = input.trim();

let reversed = reverse\_string(input);

println!("The reversed string is: {}", reversed);

}

1. Check if a number is prime in Rust.

use std::io;

fn is\_prime(num: u64) -> bool {

if num <= 1 {

return false;

}

for i in 2..=num / 2 {

if num % i == 0 {

return false;

}

}

true

}

fn main() {

println!("Enter a number:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let num: u64 = input.trim().parse().expect("Invalid input");

if is\_prime(num) {

println!("{} is a prime number.", num);

} else {

println!("{} is not a prime number.", num);

}

}

1. Merge two sorted arrays in Rust.

use std::io;

fn merge\_sorted\_arrays(arr1: &[i32], arr2: &[i32]) -> Vec<i32> {

let mut merged = Vec::new();

let (mut i, mut j) = (0, 0);

while i < arr1.len() && j < arr2.len() {

if arr1[i] <= arr2[j] {

merged.push(arr1[i]);

i += 1;

} else {

merged.push(arr2[j]);

j += 1;

}

}

while i < arr1.len() {

merged.push(arr1[i]);

i += 1;

}

while j < arr2.len() {

merged.push(arr2[j]);

j += 1;

}

merged

}

fn main() {

println!("Enter the first sorted array of integers separated by spaces:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let arr1: Vec<i32> = input

.split\_whitespace()

.filter\_map(|s| s.parse().ok())

.collect();

println!("Enter the second sorted array of integers separated by spaces:");

input.clear();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let arr2: Vec<i32> = input

.split\_whitespace()

.filter\_map(|s| s.parse().ok())

.collect();

let merged = merge\_sorted\_arrays(&arr1, &arr2);

println!("Merged sorted array: {:?}", merged);

}

1. Find the maximum subarray sum in Rust.

use std::io;

fn max\_subarray\_sum(arr: &[i32]) -> i32 {

let mut max\_sum = arr[0];

let mut current\_sum = arr[0];

for &num in arr.iter().skip(1) {

current\_sum = current\_sum.max(num);

max\_sum = max\_sum.max(current\_sum);

}

max\_sum

}

fn main() {

println!("Enter the array of integers separated by spaces:");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line");

let arr: Vec<i32> = input

.split\_whitespace()

.filter\_map(|s| s.parse().ok())

.collect();

let max\_sum = max\_subarray\_sum(&arr);

println!("The maximum subarray sum is: {}", max\_sum);

}