Internship Test

Rust Developer Profile Set-1

Instructions:

- You have to solve the below questions using **Rust.**
- You have **1 day** to submit this.
- To submit your code, upload it to github and fill the below form

  https://docs.google.com/forms/d/e/1FAIpQLSeQ-9PffLbFkzTFfNNv6SqmlyKhV8OT5TJVkPiHBOq9G1-YTQ/viewform

Questions:

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

```
use std::io;


fn is_palindrome(input: &str) -> bool {
    let input = input.to_lowercase();
    let reversed = input.chars().rev().collect::<String>();
    input == 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 trailing newline



    if is_palindrome(input) {

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

    } else {

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

    }

}
```

2. 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> {
    let mut low = 0;
    let mut high = arr.len() - 1;
    let mut result: Option<usize> = None;

    while low <= high {
        let mid = low + (high - low) / 2;

        if arr[mid] == target {
            result = Some(mid);
            high = mid - 1; // Look for the first occurrence on the left side
        } else if arr[mid] < target {
            low = mid + 1;
        } else {
            high = mid - 1;
```

```
            }
        }

        result
    }

    fn main() {
        println!("Enter sorted 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
            .trim()
            .split_whitespace()
            .map(|s| s.parse().expect("Invalid input"))
            .collect();

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

        let mut target_input = String::new();
        io::stdin().read_line(&mut target_input).expect("Failed to read line");

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

        if let Some(index) = first_occurrence_index(&arr, target) {
            println!("The first occurrence of {} is at index {}", target, index);
        } else {
            println!("{} is not found in the array.", target);
        }
    }
```

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

```
use std::io;

fn shortest_word(input: &str) -> Option<&str> {
    input.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 shortest = shortest_word(&input.trim());

    match shortest {
```

```
            Some(word) => println!("The shortest word is: {}", word),
            None => println!("No words found in the input."),
        }
    }
```

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

```
use std::io;

fn is_prime(n: u64) -> bool {
    if n <= 1 {
        return false;
    }
    if n <= 3 {
        return true;
    }
    if n % 2 == 0 || n % 3 == 0 {
        return false;
    }
    let mut i = 5;
    while i * i <= n {
        if n % i == 0 || n % (i + 2) == 0 {
            return false;
        }
        i += 6;
    }
    true
}

fn main() {
    println!("Enter a number:");

    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");

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

    if is_prime(number) {
        println!("{} is a prime number.", number);
    } else {
        println!("{} is not a prime number.", number);
    }
}
```

5. 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] as f64 + arr[mid] as f64) / 2.0
    } else {
        arr[len / 2] as f64
    }
}

fn main() {
    println!("Enter sorted 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
        .trim()
        .split_whitespace()
        .map(|s| s.parse().expect("Invalid input"))
        .collect();

    let median_value = median(&arr);

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

6. 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];
    let mut prefix = String::new();
    'outer: for (i, ch) in first_string.chars().enumerate() {
        for string in &strings[1..] {
            if let Some(c) = string.chars().nth(i) {
                if c != ch {
                    break 'outer;
                }
            } else {
                break 'outer;
            }
        }
        prefix.push(ch);
    }
    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
        .trim()
        .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);
    }
}
```

7. 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 == 0 || k > arr.len() {
        return None; // Invalid k
    }
    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
        .trim()
        .split_whitespace()
        .map(|s| s.parse().expect("Invalid input"))
        .collect();

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

    let mut k_input = String::new();
    io::stdin().read_line(&mut k_input).expect("Failed to read line");
```

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

    match kth_smallest(&arr, k) {
        Some(value) => println!("The {}th smallest element is: {}", k, value),
        None => println!("Invalid value of k."),
    }
}
```

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

```
use std::io;

// Define the binary tree structure
#[derive(Debug)]
struct TreeNode {
    val: i32,
    left: Option<Box<TreeNode>>,
    right: Option<Box<TreeNode>>,
}

impl TreeNode {
    // Constructor for TreeNode
    fn new(val: i32) -> Self {
        TreeNode { val, left: None, right: None }
    }
}

fn construct_tree() -> Option<Box<TreeNode>> {
    println!("Enter the value of the root node:");
    let root_val: i32 = read_input();

    let mut root = Some(Box::new(TreeNode::new(root_val)));
    let mut queue = std::collections::VecDeque::new();
    queue.push_back(&mut root);

    while let Some(node) = queue.pop_front() {
        let node = match node {
            Some(n) => n,
            None => continue,
        };

        println!("Enter the value of the left child of node {} (or enter -1 for no child):",
node.val);
        let left_val: i32 = read_input();
        if left_val != -1 {
            let left_child = Some(Box::new(TreeNode::new(left_val)));
            node.left = left_child;
            queue.push_back(&mut node.left);
        }
```

```
        println!("Enter the value of the right child of node {} (or enter -1 for no child):",
node.val);
        let right_val: i32 = read_input();
        if right_val != -1 {
            let right_child = Some(Box::new(TreeNode::new(right_val)));
            node.right = right_child;
            queue.push_back(&mut node.right);
        }
    }

    root
}

fn max_depth(root: &Option<Box<TreeNode>>) -> i32 {
    match root {
        Some(node) => {
            let left_depth = max_depth(&node.left);
            let right_depth = max_depth(&node.right);
            1 + left_depth.max(right_depth)
        }
        None => 0,
    }
}

fn read_input() -> i32 {
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");
    input.trim().parse().expect("Invalid input")
}

fn main() {
    println!("Construct the binary tree:");
    let root = construct_tree();
    let depth = max_depth(&root);
    println!("The maximum depth of the binary tree is: {}", depth);
}
```

9. Reverse a string in Rust

```
use std::io;

fn main() {
    println!("Enter a string to reverse:");

    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");

    let reversed = reverse_string(&input.trim());

    println!("Reversed string: {}", reversed);
}
```

```
fn reverse_string(input: &str) -> String {
    input.chars().rev().collect()
}
```

10. Check if a number is prime in Rust

```
use std::io;

fn is_prime(n: u64) -> bool {
    if n <= 1 {
        return false;
    }
    if n <= 3 {
        return true;
    }
    if n % 2 == 0 || n % 3 == 0 {
        return false;
    }
    let mut i = 5;
    while i * i <= n {
        if n % i == 0 || n % (i + 2) == 0 {
            return false;
        }
        i += 6;
    }
    true
}

fn main() {
    println!("Enter a number:");

    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");

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

    if is_prime(number) {
        println!("{} is a prime number.", number);
    } else {
        println!("{} is not a prime number.", number);
    }
}
```

11. Merge two sorted arrays in Rust

```
use std::io;

fn merge_sorted_arrays(arr1: &[i32], arr2: &[i32]) -> Vec<i32> {
    let mut merged = Vec::with_capacity(arr1.len() + arr2.len());
    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;
        }
    }

    // Add remaining elements from arr1
    while i < arr1.len() {
        merged.push(arr1[i]);
        i += 1;
    }

    // Add remaining elements from arr2
    while j < arr2.len() {
        merged.push(arr2[j]);
        j += 1;
    }

    merged
}

fn main() {
    println!("Enter sorted array 1 separated by spaces:");
    let mut input1 = String::new();
    io::stdin().read_line(&mut input1).expect("Failed to read line");
    let arr1: Vec<i32> = input1.trim().split_whitespace()
        .map(|s| s.parse().expect("Invalid input"))
        .collect();

    println!("Enter sorted array 2 separated by spaces:");
    let mut input2 = String::new();
    io::stdin().read_line(&mut input2).expect("Failed to read line");
    let arr2: Vec<i32> = input2.trim().split_whitespace()
        .map(|s| s.parse().expect("Invalid input"))
        .collect();

    let merged = merge_sorted_arrays(&arr1, &arr2);
    println!("Merged sorted array: {:?}", merged);
}
```

12. Find the maximum subarray sum in Rust

```
use std::io;
```

```
fn max_subarray_sum(arr: &[i32]) -> i32 {
    let mut max_sum = 0;
    let mut current_sum = 0;

    for &num in arr {
        current_sum = current_sum + num;
        if current_sum < 0 {
            current_sum = 0;
        }
        if max_sum < current_sum {
            max_sum = 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.trim().split_whitespace()
```

```
            .map(|s| s.parse().expect("Invalid input"))
            .collect();

    let max_sum = max_subarray_sum(&arr);
    println!("Maximum subarray sum: {}", max_sum);
}
```

CONTACT

Please feel free to contact us in case you have any questions regarding anything about this internship at **hr.quadbtech@gmail.com.** Please put “React JS Internship” as the email subject so that we can filter out your submission/question before anything else.