QuadBTECH ASSIGNMENT

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

Code:

```
fn is_palindrome(s: &str) -> bool {
    let mut s_chars = s.chars().filter(|c| c.is_alphanumeric());
    while let (Some(left), Some(right)) = (s_chars.next(), s_chars.next_back()) {
        if left.to_ascii_lowercase() != right.to_ascii_lowercase() {
            return false;
        }
    }
    true
}

fn main() {
    let input = "A man, a plan, a canal, Panama!";
    println!("Is '{}' a palindrome? {}", input, is_palindrome(input));
}
```

Output:

```
Is 'A man, a plan, a canal, Panama!' a palindrome? true


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
fn first_occurrence_index(nums: &[i32], target: i32) -> Option<usize> {
    let mut low = 0;
    let mut high = nums.len() - 1;

    while low <= high {
        let mid = low + (high - low) / 2;
        if nums[mid] == target && (mid == 0 || nums[mid - 1] != target) {
            return Some(mid);
        } else if nums[mid] < target {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    None
}

fn main() {
    let nums = vec![1, 2, 2, 3, 4, 4, 4, 5, 6];
    let target = 4;
    if let Some(index) = first_occurrence_index(&nums, target) {
        println!("First occurrence of {} is at index {}", target, index);
    } else {
        println!("{} is not present in the array", target);
    }
}
```

Output:

```
First occurrence of 4 is at index 4


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code**:**

```
fn shortest_word(s: &str) -> Option<&str> {
   s.split_whitespace().min_by_key(|word| word.len())
}

fn main() {
   let sentence = "The quick brown fox jumps over the lazy dog";
   if let Some(shortest) = shortest_word(sentence) {
      println!("The shortest word is: {}", shortest);
   } else {
      println!("The input string is empty");
   }
}
```

Output:

```
The shortest word is: The


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
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() {
    let number = 17;
    if is_prime(number) {
        println!("{} is a prime number.", number);
    } else {
        println!("{} is not a prime number.", number);
```

```
    }
}
```

Output:

```
17 is a prime number.


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
fn find_median(nums: &[i32]) -> f64 {
    let n = nums.len();
    if n % 2 == 0 {
        let mid_right = n / 2;
        let mid_left = mid_right - 1;
        return (nums[mid_left] + nums[mid_right]) as f64 / 2.0;
    } else {
        return nums[n / 2] as f64;
    }
}

fn main() {
    let nums = vec![3, 6, 7, 8, 10, 12];
    println!("Median: {}", find_median(&nums));

    let nums = vec![1, 2, 4, 5, 9];
    println!("Median: {}", find_median(&nums));
}
```

Output:

```
Median: 7.5
Median: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
fn longest_common_prefix(strs: &[String]) -> String {
    if strs.is_empty() {
        return String::new();
    }

    let mut prefix = strs[0].clone();

    for s in strs.iter().skip(1) {
        while !s.starts_with(&prefix) {
            prefix.pop();
            if prefix.is_empty() {
                return String::new();
            }
        }
    }

    prefix
}

fn main() {
    let strings = vec![
        String::from("flower"),
        String::from("flow"),
        String::from("flight"),
    ];
    println!("Longest Common Prefix: {}", longest_common_prefix(&strings));
}
```

Output:

```
Longest Common Prefix: fl


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
fn kth_smallest(nums: &[i32], k: usize) -> Option<i32> {
    if k > nums.len() {
        return None;
    }

    let mut sorted_nums = nums.to_vec();
    sorted_nums.sort();

    Some(sorted_nums[k - 1])
}

fn main() {
    let nums = vec![7, 10, 4, 3, 20, 15];
    let k = 3;

    if let Some(kth) = kth_smallest(&nums, k) {
        println!("The {}th smallest element is: {}", k, kth);
    } else {
        println!("Invalid input.");
    }
}
```

Output:

```
The 3th smallest element is: 7


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
#[derive(Debug, PartialEq, Eq)]
pub struct TreeNode {
    pub val: i32,
    pub left: Option<Box<TreeNode>>,
    pub right: Option<Box<TreeNode>>,
}

impl TreeNode {
    #[inline]
    pub 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);
            let right_depth = max_depth(node.right);
            1 + left_depth.max(right_depth)
        },
        None => 0,
    }
}

fn main() {
    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))),
        })),
    }));
```

```
    println!("Maximum depth of the tree: {}", max_depth(root));
}
```

Output:

```
Maximum depth of the tree: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

9. **Reverse a string in Rust**

Code:

```
fn reverse_string(s: &str) -> String {
    let mut reversed = String::new();
    for c in s.chars().rev() {
        reversed.push(c);
    }
    reversed
}

fn main() {
    let original = "Hello, world!";
    let reversed = reverse_string(original);
    println!("Original: {}", original);
    println!("Reversed: {}", reversed);
}
```

Output:

```
Original: Hello, world!
Reversed: !dlrow ,olleH


...Program finished with exit code 0
Press ENTER to exit console.
```

10. Check if a number is prime in Rust

Code:

```
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() {
    let number = 19;
    if is_prime(number) {
        println!("{} is a prime number.", number);
    } else {
        println!("{} is not a prime number.", number);
    }
}
```

Output:

```
19 is a prime number.


...Program finished with exit code 0
Press ENTER to exit console.
```

11. Merge two sorted arrays in Rust

Code:

```
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() {
```

```
    let arr1 = vec![1, 3, 5, 7];
    let arr2 = vec![2, 4, 6, 8];

    let merged = merge_sorted_arrays(&arr1, &arr2);

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

Output:

```
Merged array: [1, 2, 3, 4, 5, 6, 7, 8]

...Program finished with exit code 0
Press ENTER to exit console.
```

12. Find the maximum subarray sum in Rust

Code:

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

    for &num in nums.iter().skip(1) {
        current_sum = current_sum.max(num);
        max_sum = max_sum.max(current_sum);
    }

    max_sum
}

fn main() {
    let nums = vec![-2, 1, -3, 4, -1, 2, 1, -5, 4];
    println!("Maximum subarray sum: {}", max_subarray_sum(&nums));
}
```

Output:

```
Maximum subarray sum: 4

...Program finished with exit code 0
Press ENTER to exit console.
```