Rust Lab Exercise – Make-up 13/10/2025

No on-line devices allow, only paper book.

Please write the Rust code for:

1. (10 Points) Create a command-line for a word-searching tool which demonstrates the concept of closures and CLI in Rust. To be precise, please design a program that **searches** for words in a **text file** using closures to handle the search logic. The program will contain:

- **CLI Component**:
  - **accept two arguments**: file_name and search_word,
  - read file contents and process them line by line, and
  - display results showing **total matches** and **line numbers**
- **Closure Component**:
  - Create a closure that **captures the search_word**
  - Use this closure to process each line of the file
  - The closure tracks both match count and line numbers

Sample Run of the program:

```
./word_finder example.txt Rust
Found 3 occurrences of "Rust" at lines: 1, 4, 7
```

---

2. (10 points) Assume there is a **cheetah** (lives on grassland/savannah, carnivore) and a **horse** (lives on grassland/pasture, herbivore). They want to race each other. Write a Rust program to simulate this race. Each animal has distinct movement speed, habitat, and diet.

4.1) Trait

Create a trait Animal with:

- move_forward(&self) -> f32 — returns forward distance per tick (meters) based on speed.
- get_name(&self) -> &str
- get_habitat(&self) -> &str
- get_diet(&self) -> &str

4.2) Structs

Create two structs:

- Cheetah { speed_mps: f32, habitat: String, diet: String, name: String }
- Horse { speed_mps: f32, habitat: String, diet: String, name: String }

4.3) Implement Trait

- Cheetah::move_forward() returns speed_mps * 1.5
- Horse::move_forward() returns speed_mps * 0.5

4.4) Race Simulation

Write fn race(a: &dyn Animal, b: &dyn Animal, distance: f32) -> &str

Simulate tick-by-tick until one (or both) reaches distance meters. Return the winner's name (or "Tie" if both cross in the same tick).

4.5) Main

Instantiate a cheetah and a horse with speeds, habitats, diets, print each animal's habitat & diet, call race, and print the winner.

Code and scoring
Problem 1

```
fn main() {
    // Collect command-line arguments
    let args: Vec<String> = env::args().collect();                    (1)

    // Ensure we have the correct number of arguments
    if args.len() != 3 {                                    (1)
        eprintln!("Usage: {} <file_name> <search_word>", args[0]);
        process::exit(1);
    }

    let file_name = &args[1];                               (0.25)
    let search_word = &args[2];                             (0.25)

    // Attempt to open the file
    let file = File::open(file_name);                       (0.5)
    let file = match file {
        Ok(file) => file,                                           (0.25)
        Err(err) => {                                               (0.25)
            eprintln!("Error opening file {}: {}", file_name, err);  (0.25)
            process::exit(1);
        }
    };

    // Use a closure to capture the search_word and perform the search
    let mut match_count = 0;                                (0.25)
    let mut line_numbers = vec![];                          (0.25)

    // Closure to process each line
    let search = |line: &str, line_number: usize| {             (1)
        if line.contains(search_word) {                         (0.5)
            match_count += 1;                               (0.5)
            line_numbers.push(line_number);                 (0.5)
        }
```

```
    };

    // Read and process each line in the file
    let reader = io::BufReader::new(file);              (0.5)
    for (index, line) in reader.lines().enumerate() {         (0.5)
        if let Ok(content) = line {
            search(&content, index + 1); // Pass line and line number to the
closure (1)
        }
    }

    // Display results
    if match_count > 0 {                                (0.5)
        println!(
            "Found {} occurrences of \"{}\" at lines: {:?}",         (0.5)
            match_count, search_word, line_numbers
        );
    } else {                                                    (0.25)
        println!("No occurrences of \"{}\" found in the file.", search_word);
    }
}
```

Problem 2

```
// [Score 1.0] Define the Animal trait with required methods.
trait Animal {
    fn move_forward(&self) -> f32; // Distance covered in a forward move
    fn get_name(&self) -> &str;
    fn get_habitat(&self) -> &str;
    fn get_diet(&self) -> &str;
}

// [Score 0.5] Define the Cheetah struct (fields: name, speed, habitat, diet).
struct Cheetah {
    name: String,
    speed: f32,      // meters per second
    habitat: String,
    diet: String,
}

// [Score 0.5] Define the Horse struct (fields: name, speed, habitat, diet).
struct Horse {
    name: String,
    speed: f32,      // meters per second
```

```
    habitat: String,
    diet: String,
}

// [Score 1.5] Implement Animal for Cheetah (move = speed * 1.5 + getters).
impl Animal for Cheetah {
    fn move_forward(&self) -> f32 {
        self.speed * 1.5 // Cheetah moves 1.5x its speed per move
    }
    fn get_name(&self) -> &str { &self.name }
    fn get_habitat(&self) -> &str { &self.habitat }
    fn get_diet(&self) -> &str { &self.diet }
}

// [Score 1.5] Implement Animal for Horse (move = speed * 0.5 + getters).
impl Animal for Horse {
    fn move_forward(&self) -> f32 {
        self.speed * 0.5 // Horse moves 0.5x its speed per move
    }
    fn get_name(&self) -> &str { &self.name }
    fn get_habitat(&self) -> &str { &self.habitat }
    fn get_diet(&self) -> &str { &self.diet }
}

// [Score 2.5] Race function using &dyn Animal (signature, loop simulation,
winner logic).
fn race(a: &dyn Animal, b: &dyn Animal, distance: f32) -> String {
    let mut pa = 0.0f32;
    let mut pb = 0.0f32;

    while pa < distance && pb < distance {
        pa += a.move_forward();
        pb += b.move_forward();
    }

    if pa >= distance && pb >= distance {
        "It's a tie!".to_string()
    } else if pa >= distance {
        a.get_name().to_string()
    } else {
        b.get_name().to_string()
    }
}

fn main() {
    // [Score 0.75] Create the Cheetah instance with proper fields.
    let cheetah = Cheetah {
        name: "Cheetah".to_string(),
        speed: 28.0, // meters per second (illustrative)
        habitat: "Savannah/Grasslands".to_string(),
        diet: "Carnivore".to_string(),
    };
```

```
    // [Score 0.75] Create the Horse instance with proper fields.
    let horse = Horse {
        name: "Horse".to_string(),
        speed: 15.0, // meters per second (illustrative)
        habitat: "Grassland/Pasture".to_string(),
        diet: "Herbivore".to_string(),
    };

    // [Score 0.5] Print each animal's habitat and diet.
    println!("{} lives in {} and eats {}.", cheetah.get_name(),
cheetah.get_habitat(), cheetah.get_diet());
    println!("{} lives in {} and eats {}.", horse.get_name(),
horse.get_habitat(), horse.get_diet());

    // [Score 0.25] Simulate the race by calling race(&dyn Animal, &dyn Animal,
distance).
    let race_distance = 500.0; // meters
    let winner = race(&cheetah, &horse, race_distance);

    // [Score 0.25] Announce the winner (or tie).
    println!("The winner is: {}", winner);
}
```