**✅ Homework 4: Rust Social Sent Scores – Master Objective List**

**🧠 Q1: Written Response Objectives (10 points total)**

Answer each question using what you’ve learned about Rust, supported by your course notes and files (we’ll draft these together later if you want):

1. **Memory Allocation**
   * Manual vs. Garbage Collected vs. Ownership (Rust uses Ownership).
   * Talk about memory safety, dangling pointers, performance.
   * Rust has a unique way of handling memory allocation. They are not dynamically allocated nor freed. It uses a heap as a way of allocating lifetime depending on the box values pointing to it. This may outlive the frame they are allocated within. Yet it is statically stored in the heap as the position will not change even if the t box value is moved.
2. **Type System & Binding Time**
   * Rust is statically typed. It must know the types of all variables at compile time. The compiler can infer what type we want to use based on the value and how it is used.
   * Discuss compile-time checks vs. runtime flexibility.
3. **Subprogram Calling Conventions**
   * Parameter passing (by value, by reference &T, or mutable &mut T).
   * Return via expression or return.
   * Mention stack frames and first-class functions.
4. **Variable Scope and Lifetime**
   * Explain lexical scoping, lifetimes, and Rust's borrow checker.
   * Use recursion or closures to illustrate.
5. **Unique Rust Feature**
   * Ownership/Borrowing, Pattern Matching, or Macros.
   * Discuss advantages, impact on design and structure.

**🧑‍💻 Q2: Programming Assignment in Rust (40 points)**

**✅ Phase 1: Setup**

* **Create Rust project:**
* cargo new social\_sent\_score
* cd social\_sent\_score
* **File structure:**
* src/
* main.rs
* sentiment.rs (for helper subprograms)
* data/
* socialsent.csv
* good.txt / bad.txt / review.txt

**✅ Phase 2: Build Subprograms**

1. **build\_social\_sentiment\_table(file\_path: &str) -> HashMap<String, f64>**
   * Reads the CSV and returns HashMap<String, f64>.
   * Use std::fs and csv crate or split().
2. **get\_social\_sentiment\_score(review\_file: &str, sentiment\_table: &HashMap<String, f64>) -> f64**
   * Loops through words in the review file, sums the sentiment values.
   * Print: [word: current\_score, accumulated\_score].
3. **get\_star\_rating(score: f64) -> u8**
   * Uses Rust’s match or if-else to map final score to stars:
   * match score {
   * s if s < -5.0 => 1,
   * s if s < -1.0 => 2,
   * s if s < 1.0 => 3,
   * s if s < 5.0 => 4,
   * \_ => 5
   * }

**✅ Phase 3: CLI Support**

* Accept review file as command-line argument (or use "review.txt" by default):
* let args: Vec<String> = env::args().collect();
* let review\_file = args.get(1).map(String::as\_str).unwrap\_or("review.txt");

**✅ Phase 4: Output & Reporting**

* Print each [word: current\_score, accumulated\_score] line.
* Final print:
* <file>.txt score: X.XX
* <file>.txt Stars: Y

**✅ Phase 5: Documentation & Submission**

* **Markdown/PDF Report**
  + Abstract (overview of Rust)
  + Approach (design, CSV parsing, scoring, star rating)
  + New things you learned (e.g., ownership, HashMap, CLI handling)
  + Likes/Dislikes of Rust
  + Notes (e.g., if you used ChatGPT, GitHub Copilot, etc.)
* **README.md**
  + How to compile and run:
  + cargo build
  + cargo run -- good.txt
* **Output Files**
  + Save example outputs (copy-paste into .txt files)
* **Submit on Canvas**
  + Either upload your full zipped Rust project, or link to REPL with:
    - main.rs, Cargo.toml, README.md, output.txt

Let me know when you're ready to:

1. Start writing Q1 answers.
2. Begin coding part of the Rust app.
3. Get a sample socialsent.csv or review.txt to test with.

We’ll do this one milestone at a time. What would you like to start with?