

## Rust Lab 12 – Rust CLI and File Management

24/9/2025

### Lab 1: Argument Count Validation

Create a Rust program that:

- Create a Rust program that accepts command-line arguments using `std::env::args`.
- Check if exactly 3 arguments are passed (excluding program name).
- If correct, print each argument on a single line.
- If incorrect, display an error message with argument count and exit with a nonzero code using `std::process::exit`.
- Use `Result` for error handling. Write `main` as `fn main() -> Result<(), Box<dyn std::error::Error>>`.

Sample output: Command: `./program arg1 arg2 arg3`

Output: `Arguments provided: arg1, arg2, arg3`

Command: `./program arg1`

Output: Error: `Exactly 3 arguments are required. You provided 1.`

TA Check: \_\_\_\_\_

### Lab 2: Command-Line Calculator

Create a Rust program that:

- Take 3 command-line arguments: two numbers and one operator (+, -, \*, /).
- Parse both numbers using `.parse::<T>()`, and exit with an error if parsing fails.
- Validate the operator; exit with error for unsupported symbols.
- Perform the calculation, handling division by zero as an error.
- Use error propagation and provide clear error messages.
- Exit with status 1 on user or runtime errors.
- Handles potential errors such as:
  - Missing arguments
  - Invalid numbers (non-numeric input)
  - Unsupported operators (anything other than +, -, \*, /)
  - Division by zero

TA Check: \_\_\_\_\_

### Lab 3: Line, Word, and Character Counter for a Text File

Create a Rust program that:

- Exit with status 1 on user or runtime errors.
- Accepts a file path as a command-line argument.
- Reads the contents of the file and counts the number of lines, words, and characters.
- Displays these counts to the user.
- Handles errors such as:
  - Missing file path argument
  - File not found or unreadable

Sample output: File: sample.txt

Lines: 10

Words: 50

Characters: 350

TA Check: \_\_\_\_\_

---

## Lab 4: Console/Text File I/O & Environment Vars

Create a Rust program that:

- Prompt the user to enter multiple lines of text (end on empty line).
- **Before saving:**
  - Read and write these three system values at the top of the file:
    - **OS** (e.g., using the `std::env::consts::OS`)
    - **User** (e.g., via the `USER` or `USERNAME` environment variable using `std::env::var`)
- After, append all user input lines as normal.
- Re-open and display the file contents in uppercase (buffered).

**Hint for System Environment:**

- Use `std::env::consts::OS` for operating system name.
- Use `std::env::var("USER")` (on Unix) or `std::env::var("USERNAME")` (on Windows) to get the user.

TA Check: \_\_\_\_\_