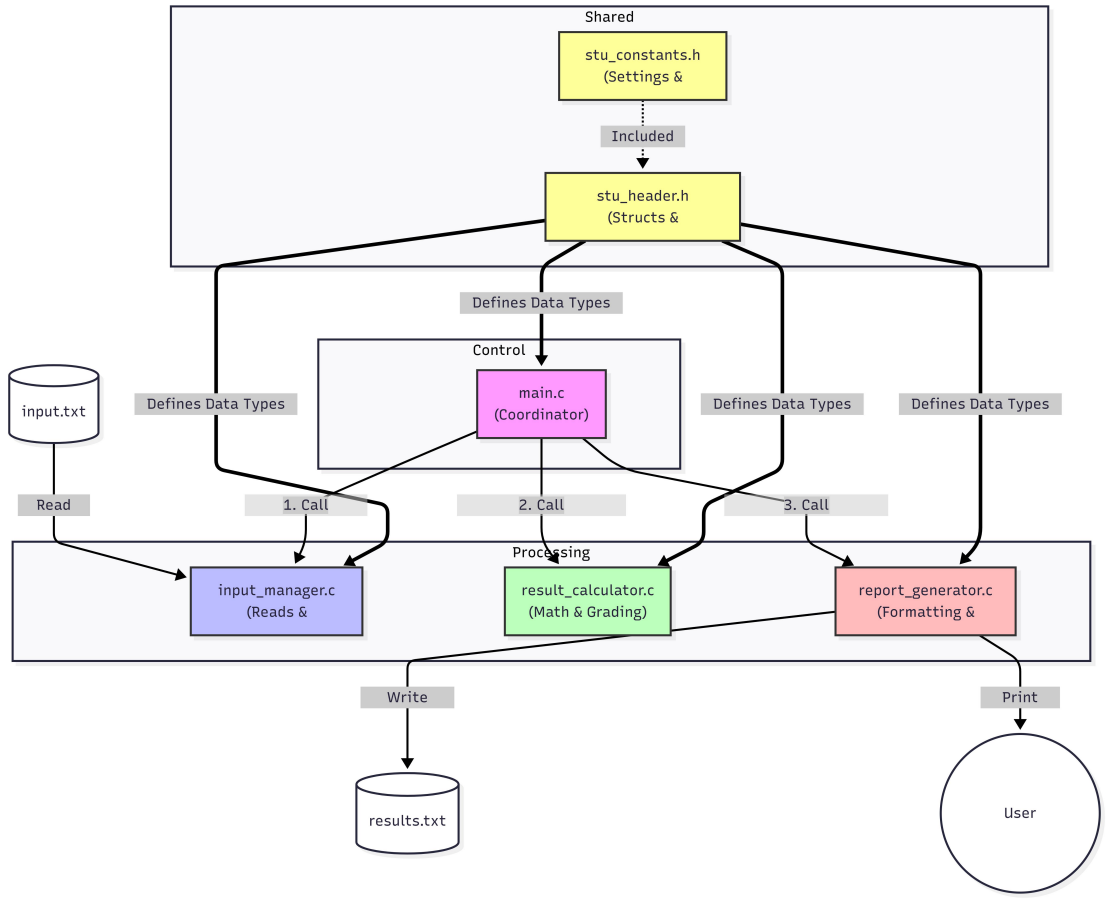
Name: Aditya Kumar
Reg. No.: 25MCMT10
Documentation of SE LAB 2

Software Architecture Diagram



1. System Overview

The Student Result Processing System is a modular C application designed to manage student academic records. It reads raw data from a file, validates it for integrity, performs calculations (Total, Percentage, Grade, CGPA), and generates a formatted report for both the console and a permanent output file.

The system is architecture into four distinct modules to ensure high cohesion (each module does one thing well) and low coupling (modules rely on shared data structures rather than each other's internal logic).

2. System Architecture & Coordination

The program follows a strict **Input-Process-Output (IPO)** workflow, coordinated by the **Main Driver.**

How the Modules Coordinate:

1) main.c acts as the Coordinator. It owns the memory (the Student array) but performs no work itself.
2) stu_constants.h acts as the Configuration Center. It defines the rules (e.g., Passing Mark = 50) that all modules must obey.
3) input_manager.c fills the memory with data from the disk.
4) result_calculator.c modifies the memory by adding grades and totals.
5) report_generator.c reads the memory and writes the final report to disk.

3. Module Specifications

Module 1: Input Manager (input_manager.c)

- **Role:** Reads raw data and filters out invalid records.
- **Input:** input.txt (File), Empty Student Array (Memory).
- **Pre-condition:** Input file must exist.
- **Logic (Algorithm):**
  1. Open input.txt. If NULL, return error.
  2. Loop through each line:
     - ❖ Read ID, Name.
     - ❖ **Validate ID:** Check if alphanumeric & unique.
     - ❖ **Validate Name:** Check if alphabetic only.
     - ❖ **Validate Marks:** Loop 5 times. Check range (Minor 0-40, Major 0-60).
     - ❖ If **ANY** check fails => Set valid flag to False and print error.
     - ❖ If valid is True => Save data to Student array.
  3. Close file.
  4. Return count of valid students.
- Output: Populated Student array, Count of valid records.

Module 2: Result Calculator (result_calculator.c)

- **Role:** Performs all mathematical computations.
- **Input:** Populated Student array, Count of students.
- **Pre-condition:** Array must contain valid data.
- **Logic (Algorithm):**
  1. Loop through each student i from 0 to count.
  2. Initialize GrandTotal = 0, FailFlag = False.
  3. Loop through 5 subjects:
     - ✧ Total = Minor + Major.
     - ✧ Add to GrandTotal.
     - ✧ Check Pass Criteria: If Total < PASS_MARK (50), set FailFlag = True.
- Calculate Percentage = GrandTotal / 5.
- **Determine Grade:**
  1. If FailFlag is True => Grade = "F", CGPA = 0.0.
  2. Else map Percentage to Grade (e.g., >=90 to O, 50-55 to D).
- Store results in the struct.

Output: Updated Student array with Totals, Grades, and CGPA.

Module 3: Report Generator (report_generator.c)

- **Role:** Formats and displays the final data.
- **Input:** Fully processed Student array.
- **Pre-condition:** Calculations must be complete.
- **Logic (Algorithm):**
  1. Open results.txt for writing.
  2. Print Table Headers (ID, Name, Marks...) to Screen and File.
  3. Loop through students:
     - Format their marks into a string "m1, m2, m3...".
     - Print formatted row to Screen and File.
     - **Track Statistics:** Update ClassTotal, MaxPerc, MinPerc, and GradeCounts.
  4. Calculate ClassAverage.
  5. Print Summary Statistics (Average, Max/Min, Grade distribution) to Screen and File.
  6. Close file.

Output: results.txt file, Console Output.

Module 4: Main Driver (main.c)

- **Role:** Entry point and workflow coordinator.
- **Logic:**

Start
|-------Define Student Array[100]
|-------Call readInput()
| |-- If returns 0 -> Exit (Error)
|------- Call calculateResults()
|--------Call generateOutput()
End