Student Result Processing System (Modular C Implementation)

This document describes the detailed specification of each module in the Student Result Processing System. Each module is explained using:

- Module Name
- Input
- Pre-condition
- Output

Module 1: Student Data Module

Module Name

`students.h`

Purpose

Defines the `Student` structure used by all other modules to store academic and computed information.

Input

- Student ID (string)
- Student Name (string)
- Minor marks (array of integers)
- Major marks (array of integers)

Pre-condition

- Constant SUBJECTS must be defined
- Header file must be included before accessing student data

Output

- A `struct Student` data type containing:
    - ID, Name
    - Minor and Major marks
    - Subject totals and grades
    - Total marks, percentage, CGPA and overall grade

Module 2: Validation Module

Module Name

`validation.c / validation.h`

Purpose

Validates student ID, name and marks read from the input file.

Function: validID(char id[])

Input: Student ID string
Pre-condition: ID must be null terminated

Logic (Algorithm):

1. For each character in ID
2. If character is not alphanumeric → return invalid
3. If all characters are valid → return valid

Output:

- 1 → Valid ID
- 0 → Invalid ID

Function: validName(char name[])

Input: Student name string

Logic (Algorithm):

1. For each character in name
2. Allow only alphabets and spaces
3. If digit or special character found → invalid

Output:

- 1 → Valid name
- 0 → Invalid name

Function: validMinor(int m)

Input: Minor marks

Logic:

- If 0 ≤ m ≤ 40 → valid
- Else → invalid

Output:

- 1 → Valid
- 0 → Invalid

Function: validMajor(int m)

Input: Major marks

Logic:

- If 0 ≤ m ≤ 60 → valid
- Else → invalid

Output:

- 1 → Valid
- 0 → Invalid

Module 3: Grading Module

Module Name

`grades.c / grades.h`

Purpose

Assigns subject grades and grade points based on marks.

Function: getGradePoint(int marks)

Input: Subject total marks

Logic (Algorithm):

1. If marks ≥ 90 → return 10
2. Else if marks ≥ 85 → return 9
3. Else if marks ≥ 75 → return 8
4. Else if marks ≥ 65 → return 7
5. Else if marks ≥ 60 → return 6
6. Else if marks ≥ 55 → return 5
7. Else if marks ≥ 50 → return 4
8. Else → return 0

Output: Integer grade point (0–10)

Function: getSubjectGrade(int marks, char grade[])

Input: Subject marks

Logic:

- Compare marks with grade ranges
- Store corresponding grade string (O, A+, A, B+, B, C, D, F)

Output:

- Grade string stored in `grade[ ]`

Module 4: File Input–Output Module

Module Name

`file.c / files.h`

Purpose

Handles all file operations: reading student data and writing processed results.

Function: readFromFile(struct Student s[], int *n)

Input:

- Input file: `data/students.txt`

Pre-condition:

- File must exist
- File must follow the specified format

Logic (Pseudocode):

```
Read n
For i = 1 to n
   Read ID and Name
   Validate ID and Name
   For j = 1 to 5
```

    Read minor and major marks
    Validate marks
End

Output:

- Student array filled with input data
- Number of students stored in n

Function: writeToFile(struct Student s[], int n)

Input: Processed student array

Logic (Algorithm):

1. Open result.txt in write mode
2. Print table header
3. For each student:
    - Print ID, Name
    - Print subject totals and grades
    - Print total, percentage, grade, CGPA
4. Call `computeClassStatistics()`
5. Close file

Output:

- Complete formatted result written to `data/result.txt`

Module 5: Result Processing & Statistics Module

Module Name

`results.c / results.h`

Purpose

Processes academic results and computes class statistics.

Function: processResults(struct Student s[], int n)

Input: Student array, number of students

Output: Updated student records with totals, grades, CGPA and percentage

Function: computeClassStatistics(struct Student s[], int n, FILE *fp)

Input: Student array, output file pointer

Logic (Algorithm):

1. Initialize sum, highest, lowest
2. For each student:
    - Add percentage to sum
    - Update highest and lowest
    - Count grades (O, A+, A, B+, B, C, D, F)
3. Print:
    - Class average percentage
    - Highest percentage
    - Lowest percentage
    - Grade distribution

Output:

- Class statistics printed in `result.txt`

Module 6: Main Driver Module

Module Name

`main.c`

Purpose

Controls the execution flow of the entire application.

Logic (Flow Algorithm):

1. Start
2. Read student data from file
3. If no records → terminate
4. Process results
5. Write results to output file
6. Display completion message
7. Stop

Output

- Executes all modules in sequence
- Generates `result.txt`

Conclusion

This modular specification clearly defines the responsibility, input, processing logic and output of each module. The design ensures:

- Low coupling
- High cohesion
- Reusability of modules
- Easy testing and maintenance