Task 2: Student Result Processing System

Module 1: student.h

Module Name: student.h
Input: None

Pre-Condition:
- The file student.h must be present in the project directory.
- All source files must include this header file using #include "student.h".

Logic:
- This module defines the structure named Student.
- It also defines a constant MAX for the maximum number of students.
- The structure contains fields for student ID, name, marks, total, percentage and grade.
- This file is shared by all other modules.

Output: Provides the Student structure and MAX constant to all modules.

Module 2: validation.c

Module Name: validation.c

Input:
- Student ID as a string
- Student name as a string
- Marks as an integer
- Student structure array
- Number of students already stored

Pre-Condition:
- The Student structure must be defined.
- Input values must be read from the input file before calling validation functions.
- The files student.h and validation.h must be included.

Logic:
- The function checkID():
- Loops through each character of the student ID.
- Checks whether each character is a letter or a digit.
- Compares the ID with previously stored IDs to avoid duplicates.
- The function checkName():

- Loops through each character of the student name.
- Checks whether each character is an alphabet.
- The function checkMarks():
- Checks whether the marks are between 0 and 100.

Output:
- Returns 1 if the input is valid.
- Returns 0 if the input is invalid.

Module 3: compute.c

Module Name: compute.c

Input:
- Student structure array
- Index of the current student
- Marks of five subjects

Pre-Condition:
- Marks must already be validated.
- The Student structure must contain valid marks.
- The files student.h and compute.h must be included.

Logic:
- The function calculate():
- Initializes a variable sum to 0.
- Adds all five subject marks.
- Stores the total in the structure.
- Calculates the percentage.
- The function assignGrade():
- Reads the percentage value.
- Uses an if else ladder to assign grade based on the percentage.

Output:
- Updates the total marks of the student.
- Updates the percentage of the student.
- Assigns the grade of the student.

Module 4: output.c

Module Name: output.c

Input:

- Student structure array
- Number of students

Pre-Condition:

- Student data must already be validated and computed.
- The files student.h and output.h must be included.

Logic:

- Prints the table header.
- Loops through all students.
- Displays ID, name, total, percentage and grade.
- Calculates the class average.
- Finds the highest percentage.
- Finds the lowest percentage.
- Counts the number of students in each grade category.

Output:

- Displays the result table.
- Displays class average, highest and lowest percentage.
- Displays grade distribution.

Module 5: main.c

Module Name: main.c

Input:Input file named students_results.txt

Pre-Condition:

- All modules and header files must be present.
- The input file must exist in the project directory.
- The file format must be correct.

Logic:

- Opens the input file.
- Reads student ID and name.
- Calls validation functions to check ID and name.
- Reads marks and validates them.
- Calls compute functions to calculate total, percentage and grade.

- Stores valid student records.
- Calls the output module to display results.

Output:
- Displays student result table.
- Displays class statistics and grade count.

Summary

This modular design divides the Student Result Processing System into independent modules. Each module performs a single task and this ensures low coupling and high cohesion. The design improves readability, maintainability and reusability of the program.