Students Grade Calculator

This C++ application enables users to input, calculate, and manage student grades efficiently. It supports dynamic handling of homework and exam scores using modern object-oriented programming principles, including the Rule of Three. Users can choose between average or median calculations for final grades and import student data from files or generate scores randomly. The program outputs well-formatted tables sorted by student name or surname, making it ideal for educational use and large datasets. All code is developed following C++ best practices and is structured for clarity, extensibility, and ease of use.

Features:

Dynamic homework assignment handling with std::vector
Calculation of final grades by average or median (user's choice)
Support for random score generation and data file import
Clean, formatted output sorted by name or surname

Full implementation of the Rule of Three in the Person class
Easily extendable and well-documented source code

Final grade is calculated as:
Final Grade = 0.4 × (Homework Average/Median) + 0.6 × Exam

Release V0.1

I implemented a program that read the necessary data entered by users, with the data variable implemented in a class along with methods for processing such data for one data point. I also implemented the "rule of three" .

The class Person included the following:

- Person's first name and surname.
- Homework and exam results.
- Final grade.

The methods implemented were:

- Constructor(s).

- Assignment-copy operator (the rule of three).
- Copy constructor (the rule of three).
- Destructor (the rule of three).
- Data input method (overloaded cin).
- Data output method (overloaded cout).
- Calculation method for the final grade based on median or average, according to the user's choice.

After completing data input, I calculated the final grades by average and presented them on the screen in a format similar to the following (where the final calculated grade was presented with a precision of two decimal places):
text

```
Name       Surname       Final_Point(Aver.)
-----------------------------------------
Name1      Surname1              x.xx
Name2      Surname2              y.yy
```

I extended the program functionality to allow the calculation of the final grade using either the average or the median. The output displayed the chosen final grade calculation method clearly.

Additionally, I modified the program to handle cases where the number of homework assignments (n) was unknown in advance, enabling the user to decide when they finished entering all homework results. This part was implemented using a std::vector container to store the homework scores dynamically.

Furthermore, I added the possibility to randomly generate homework and exam scores for students within the program.

Additionally,

Created a data file named "Students.txt" with the following (preliminary) structure:

```
Name       Surname    HW1   HW2   HW3   HW4   HW5   Exam
Name1      Surname1    8     9    10     6    10     9
Name2      Surname2    7    10     8     5     4     6
```

Modified the program to read data from this file and display the calculated output as follows:

```
Name      Surname       Final (Avg.) | Final (Med.)
---------------------------------------------------
Name1     Surname1           x.xx |       y.yy
Name2     Surname2           x.xx |       y.yy
```

Output : students sorted by their names, and all columns nicely formatted.

Release V0.2:

Introduced reorganizing logic into multiple classes and header files for clarity and maintainability.

Migrated all major methods and new data types into separate header files; structured project into multiple *.cpp and *.h files.

Implemented robust exception handling for file operations and data access, including error checks and try-catch blocks for file open failures and invalid container accesses.

Enhanced random test data generation for students; generates four large test files (10,000 | 100,000 | 1,000,000 | 10,000,000 records each) with default names (e.g., Name1 Surname1).

Automated sorting and categorization of students: divides student records into "failed" (<5.0 points) and "passed" (>=5.0 points) containers.

Outputs "failed" and "passed" students into distinct data files after sort.

Added precise program speed analysis: measures execution time for each core operation (file read, sorting, splitting, and file output) using modern std::chrono timers.

All required exception management tasks and data integrity checks implemented for reliability.

Release v0.25

In this release, I performed detailed speed analysis measuring the time taken for reading data from files, sorting, splitting the list into categories, and writing output files. Five test data files with varying record counts were used for the performance measurements.

After publishing release v0.2, I created a copy of the code and adapted it to use std::list and std::deque containers instead of std::vector. I reran the speed analysis for these new container types using the same datasets (1,000; 10,000; 100,000; 1,000,000; and 10,000,000 records).

This updated code was published as release v0.25, reflecting the container flexibility and performance benchmarking enhancements.

I optimized the implementation of version V0.25 for sorting and categorizing students into "failed" and "passed" groups using three container types: std::vector, std::list, and std::deque.

Two strategies were compared:

- **Strategy 1:** Split the original student container into two new containers of the same type ("failed" and "passed"), copying students into these new containers while retaining the full original container. This approach was reviewed for memory efficiency and confirmed to be less efficient due to data duplication, but useful for performance benchmarking across container types.
- **Strategy 2:** Move "failed" students out of the original container into a separate container, removing them from the original, and then resizing the original container to contain only "passed" students. This approach improved memory usage but involved costly delete operations, especially for containers like std::vector.

Release v1.0

For each container and strategy, I precisely measured execution speed focusing on data splitting performance.

I used std::list containers predominantly and employed STL algorithms such as std::copy_if, std::remove_if, and std::back_inserter to optimize the process of sorting and categorizing students. These algorithms facilitated copying and removing elements efficiently while maintaining clean and readable code. The approach significantly improved performance by leveraging the strengths of std::list for insertion and removal operations, aligning algorithmic efficiency with container characteristics. This method was benchmarked to show measurable improvements, guiding the choice of both container and algorithm for practical use.

Application Usage Guide

1. **Preparation**
   - Ensure all source files (main.cpp, Person.cpp, Person.h, Lib.h) are in the same directory[
   - Confirm you have a C++ compiler installed (e.g., g++ on Linux/Mac or MSVC on Windows)

2. **Building with Make (Unix)**

   - Place the provided Makefile in the project directory.
   - Open a terminal and run:

```
make
```

   - This will compile the project and generate an executable

3. **Building with CMake (Cross-platform)**

   - Place the provided CMakeLists.txt in the project directory.
   - Open a terminal (or use CMake GUI), and run:

```
cmake .
cmake --build .
```

   - This will configure and build the executable for your system.

4. **Running the Application**

   - After successful compilation, run the executable:

```
./main
```

The program will:

- Generate several student files with random data, e.g. students10000.txt, students100000.txt.
- Read data from one of the student files, sort by firstname, split students based on average and median grades, and output passing students to passed.txt.
- Print times taken for key operations (reading file, sorting, splitting, output) to the console

- If there are any errors (e.g., missing student data files), they will be printed to the console