Assessment Task 5: Student Grade Analyzer

The Student Grade Analyzer is a simple command-line utility designed to manage and analyze student grades for a classroom setting. It demonstrates several fundamental programming concepts in C++, including arrays, loops, input validation, control structures, constants, and menu-driven interaction.

### **Constants**

The program defines a compile-time constant MAX\_STUDENTS to restrict the number of student entries. Constants ensure that fixed values used throughout the program are easy to maintain and prevent hard-coding.

* **Purpose**: Sets a limit on the maximum number of students (30 in this case).
* **Benefit**: Enhances code maintainability and reduces magic numbers.

### **Arrays**

Two parallel arrays are used:

* One to store student grades.
* Another to store automatically generated student IDs.
* **Concept**: Arrays are used to store multiple values of the same type in contiguous memory locations.
* **Parallel Arrays**: Multiple arrays where elements at the same index are logically related (e.g., grades[i] and studentIDs[i]).

### **Loops**

Two types of loops are used:

* **for loops**: To iterate through students for input and processing.
* **do-while loop**: To keep the menu running until the user chooses to exit.
* **Control Structure Benefit**: Keeps the application interactive and allows repeated execution of operations based on user choices.

### **Conditional Statements (if-else)**

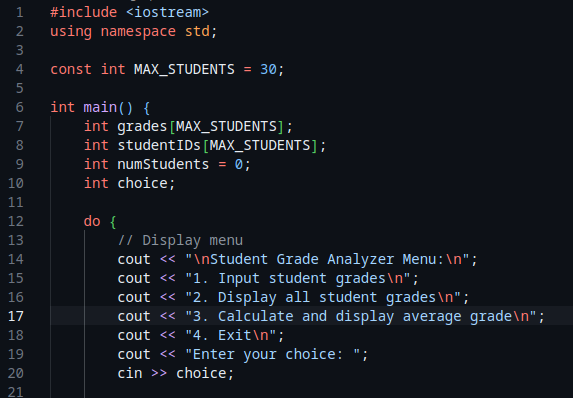
Conditional checks are employed to:

* Validate the number of students.
* Validate that grades fall within the acceptable range (0 to 100).
* Handle cases where no student data has been entered before attempting operations.
* **Role**: Provides robustness by preventing invalid or incomplete operations.

### **Switch Statement**

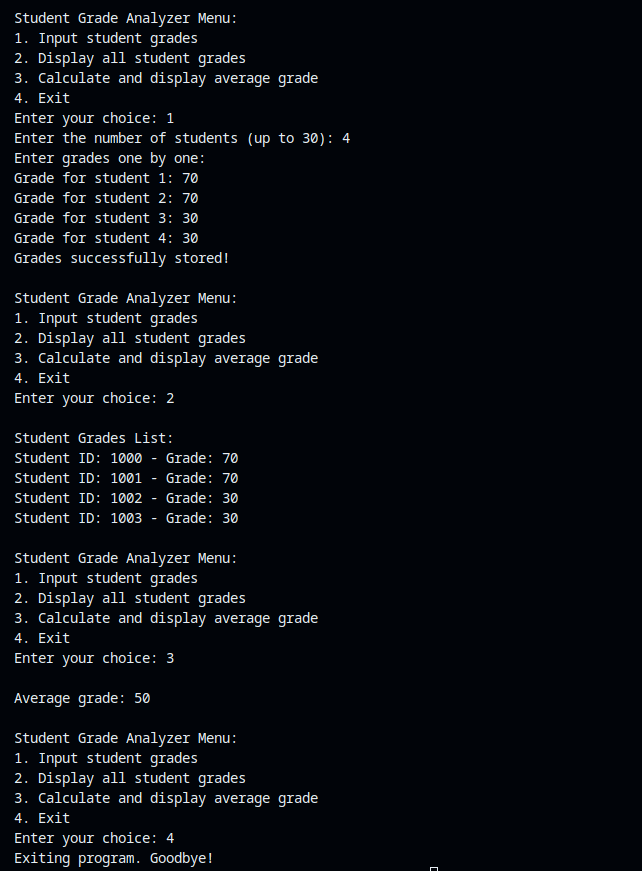
A switch statement controls the menu functionality:

* Each case represents a unique menu operation.
* A default case handles invalid menu choices.
* **Usage**: Efficiently routes user input to the corresponding functionality in a readable format.

Basic Implementation Code:  


This block initializes the variables and the display menu. Setting the maximum number of students entry allowed. It also shows the beginning of the do-while loop, inside the do loop is where the display menu is, to make the menu persistent once the code runs until the user exits the program.



This block shows the beginning of the switch statements and the end of the do-while loop and the program. The switch statements represent the various operations that the analyzer system performs based on the chosen case. And the whole block is where the program can be terminated.

This output shows the results of the various operations that the analyzer system performed.

### **Case-Insensitive Menu Input**

* **What Changed**: The program now accepts the menu choice as a char instead of an int, and uses the toupper() function to standardize the input for case-insensitive comparison.
* **Header Used**: <cctype> is included to access toupper().
* **Why It Matters**:  
  + Improves user experience by making input flexible.
  + Enhances robustness: users can enter 1, 2, 3, or 4 without worrying about capitalization or input errors (e.g., future support for 'a', 'b' type choices is easier)

### **Grade Statistics Calculation**

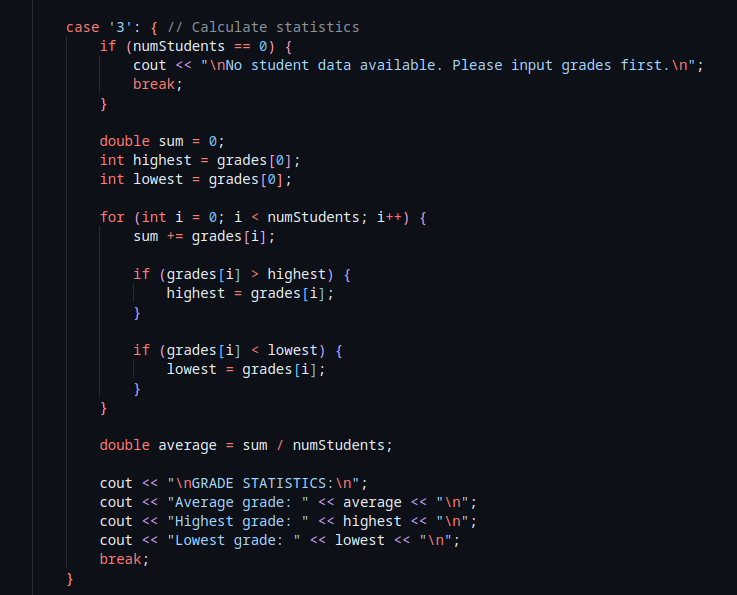
A new statistical analysis feature was added to the menu under **option 3**.

* **New Statistics Computed**:  
  + **Average Grade**: The arithmetic mean of all student grades.
  + **Highest Grade**: The maximum grade among all entries.
  + **Lowest Grade**: The minimum grade among all entries.
* **Programming Concepts Demonstrated**:  
  + **Aggregation and Iteration**: Accumulating grades for average, and comparing elements for min/max.
  + **Initialization with First Element**: highest and lowest are initialized to the first grade to ensure valid comparisons.
  + **Decision Making**: Uses if statements inside a loop to determine highest and lowest.

### **User Feedback and Interaction Improvements**

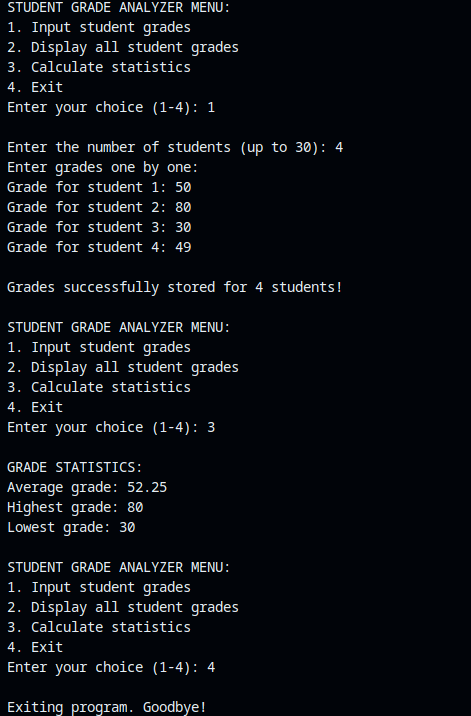
* **Enhanced Messages**: Clearer output messages now inform the user of successful operations and guide next steps.  
  + For example: "Grades successfully stored for X students!" confirms successful data entry.
  + Error messages are informative and suggest corrective action (e.g., “Please enter a value between 0 and 100”).
* **Menu Structure**: The user menu is now more consistently styled and easier to navigate.

Analyzer with statistics Code:



This code provides an extension to the basic analyzer and now includes a statistics block inside the switch statement which calculates the highest, lowest and the average grades.

Statistics output:



Now, the statistics record can be calculated after implementation of the statistics code.

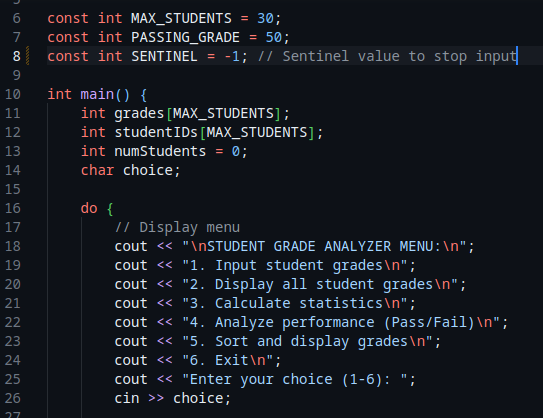
# Final Version of Student Grade Analyzer

### **Sentinel-Controlled Input Loop**

* **New Feature**: Instead of predefining the number of students, the user can now enter grades **until a sentinel value (-1)** is entered.
* This approach:  
  + Increases flexibility.
  + Allows early termination.
  + Demonstrates **sentinel-controlled loops**, a common input pattern.

### **Constants:**

Final version code:



* SENTINEL = -1 to terminate grade input.
* PASSING\_GRADE = 50 used for performance analysis.



* MAX\_STUDENTS = 30 bounds the number of grades that can be stored.

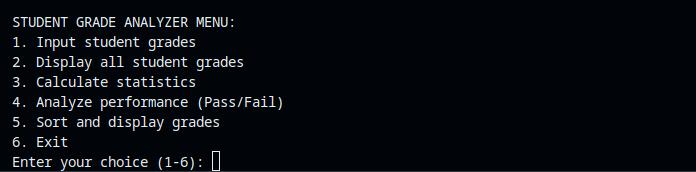
### **Performance Analysis (Pass/Fail Count)**

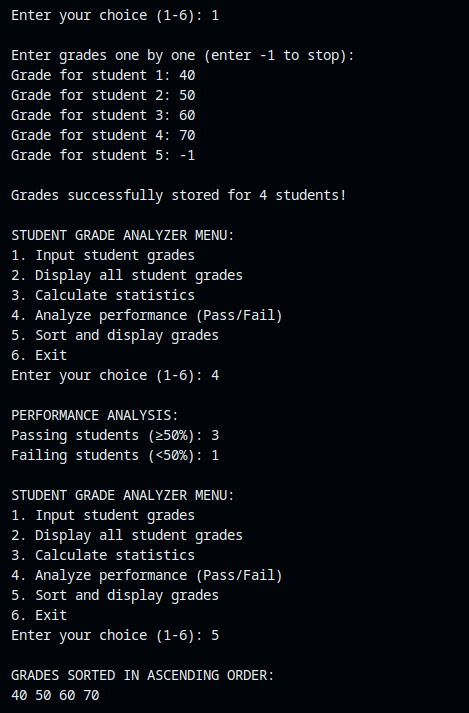
* Introduced in **option 4**.
* Separates students into **pass** and **fail** categories using the PASSING\_GRADE threshold.
* Counts and displays:  
  + Number of students passing (≥ 50%)
  + Number of students failing (< 50%)
* Demonstrates conditional filtering and counter-based aggregation.

### **Sorting Grades**

* Introduced in **option 5**.
* Uses the C++ Standard Library function std::sort() from <algorithm>.
* Operates on a **copy** of the grades array to preserve original data order.
* Shows the **use of sorting algorithms** for data presentation.

Output code:





This output shows the final implementation in its full glory. The sentinel (triggered by -1) is used to terminate input loops and go back to the main menu.   
The option 4 of analyzing the amount of grades passed/failed also implemented and output showed it is working as expected.

The option 5 also sorts the grades from lowest to highest.

Alternative implementation:

This implementation takes another direction in the implementation of the analyzer system while arriving at the same output.

The following concepts were introduced:

### **Input Buffer Management**

* **Function**: clearInputBuffer()
* Clears the leftover characters in the input stream to prevent unintended behavior.
* Especially helpful after getchar() or failed numeric input.

### **Robust Numeric Input Handling**

* Uses cin.fail() to detect when the user enters invalid input (e.g., letters instead of numbers).
* Prevents infinite input loops and ensures the program does not misbehave.

### **Use of getchar() for Menu Input**

* Avoids issues where leftover input affects subsequent reads.
* Combined with clearInputBuffer() to handle the newline character correctly.

### **Standard Algorithm Use: count\_if**

* Replaces verbose loop logic with std::count\_if() and a lambda function:

int passCount = count\_if(grades, grades + numStudents,

[](int grade) { return grade >= PASSING\_GRADE; });

### **Formatted Sorted Output**

* Sorted grades are displayed as comma-separated values:

55, 60, 71, 89

Alternative Example Code:



Output:  
