[TB14131] (IMAC-U) - 情報処理演習

C4TL1205 - LANDY Lucas (COLABS Student) - lucas.landy.t3@dc.tohoku.ac.jp

---

Table of Contents

1. Program 1: `score_sheet.c`
   - Purpose
   - Explanation of the Code
   - Execution Result
   - In-Depth Work and Demonstrations
2. Program 2: `root.c`
   - Purpose
   - Explanation of the Code
   - Execution Result
   - In-Depth Work and Demonstrations
3. Comparison and Related Concepts
4. Conclusion

Program 1: `score_sheet.c`

Purpose:

The purpose of the score_sheet.c program is to display the scores of students in three subjects, compute statistics such as the **total score**, **average**, and **deviation values**, **rank students by performance**, and **provide feedback for individual students**.

Explanation of the Code:

1. **Core Variables**:

   - `score[ID_NUM][SUB_NUM]`: A 2D array storing the scores of 10 students across 3 subjects.
   - `totalScores[]`: Stores the total scores for each student, used for ranking.
   - `subjectAvg[]` and `subjectDev[]`: Store the **average** and **standard deviation** of scores for each subject.

2. **Key Functions**:

   - `calculateAverage()`: Computes the average score for a subject.
   - `calculateDeviation()`: Computes the standard deviation of subject scores.
   - `calculateTotalAverage()`: Computes the total average score.
   - `getRank()`: Assigns ranks (A to D) based on the total score.

- `giveFeedback()`: Provides feedback for subjects where the student performed poorly (score < 70).
- `bubbleSort()`: Sorts students based on their total scores in descending order.

3. **Execution Flow**:

   1. The program calculates **total scores, averages,** and deviations for all subjects.
   2. It **sorts students** by their total scores and displays a **ranking**.
   3. Subject-wise Average: Displays the average score for each subject across all students.
   4. Averaged Total Score: Computes the overall average of total scores from all students.
   5. The user is prompted to **enter a student ID** (between 0–9).
   6. The program displays the **selected student's score sheet**, including the **total score, average, rank**, and **feedback**.
   7. It prints **deviation values** for the selected student.

Execution Result:

Input:

```
Enter student ID (0-9): 3
```

Output:

```
Student Rankings (based on total scores):
Rank  ID  Total  Rank
  1    5   290    A
  2    8   273    A
  3    6   263    B
  4    0   258    B
  5    4   249    B
  6    7   244    B
  7    2   221    C
  8    3   220    C
  9    9   208    D
 10    1   201    D

Averaged Scores for Each Subject:
SUB1: 77.50
SUB2: 85.40
SUB3: 79.80
Averaged Total Score (All Students): 242.70

Enter student ID (0-9): 3

Score Sheet for Student ID: 3
SUB1  SUB2  SUB3  Total  Average  Rank
 75    80    65    220   73.33    C
Feedback: Improve in SUB3.
```

```
Deviation values for Student ID 3:
SUB1: -0.30
SUB2: -0.58
SUB3: -1.27
```

In-Depth Work and Demonstrations:

1. **Sorting Algorithms**:
   - **Bubble Sort** was chosen for simplicity in sorting student scores. More advanced algorithms like **quick sort** could improve efficiency, especially with larger datasets.
2. **Statistical Analysis**:
   - The use of **average** and **standard deviation** provides valuable insights into student performance. These metrics help identify outliers and weak subjects.
3. **User Feedback**:
   - The program dynamically provides feedback based on each student's scores. This feature could be expanded with personalized messages for top-performing students.
4. **Possible Enhancements**:
   - **Graphical Representation**: Plot student scores and rankings using a graph for better visualization.
   - **File Input/Output**: Allow the program to read scores from a file and save results to a file.

Program 2: `root.c`

Purpose:

The purpose of the `root.c` program is to solve quadratic equations of the form:

$ax ^ 2 + b x + c = 0$

The program computes **real and complex roots** depending on the discriminant value and displays them with appropriate formatting.

Explanation of the Code:

1. **Core Variables**:
   - `a`, `b`, and `c`: Coefficients for the quadratic equation.
   - `discriminant`: The value $b ^ 2 - 4ac$ used to determine the nature of the roots.
2. **Key Functions**:
   - `calculateRoots()`: Takes between the values of a,b and c and calculates the roots.
3. **Key Logic**:

   - **Discriminant Calculation**:

     - If the discriminant is **positive**, the equation has two distinct real roots.
     - If the discriminant is **zero**, the equation has one repeated real root.
     - If the discriminant is **negative**, the equation has two complex roots.

- **Root Calculation**:
    - For real roots:

    $x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$

    - For complex roots:

    $Real Part = \frac{-b}{2a}, Imaginary Part = \frac{\sqrt{4ac - b^2}}{2a}$

Execution Result:

If the discriminant is **positive**

Input:

```
Enter coefficients a, b, and c (for ax^2 + bx + c = 0):
1 4 0.00000000000001
```

Output:

```
The roots are real and different:
Root 1 = -0.000000
Root 2 = -4.000000
```

If the discriminant is **zero**

Input:

```
Enter coefficients a, b, and c (for ax^2 + bx + c = 0):
1 -2 1
```

Output:

```
The roots are real and identical:
Root = 1.000000
```

If the discriminant is **negative**

Input:

```
Enter coefficients a, b, and c (for ax^2 + bx + c = 0):
1 2 5
```

Output:

```
The roots are complex:
Root 1 = -1.000000 + 2.000000i
Root 2 = -1.000000 - 2.000000i
```

In-Depth Work and Demonstrations:

1. **Handling Complex Roots**:
   - The program uses **complex arithmetic** to handle imaginary numbers and outputs them with the correct sign.
2. **Precision Handling**:
   - Roots are displayed up to **six decimal places** for precision. This can be extended to higher precision if needed.
3. **Edge Cases**:
   - When $a = 1, b = 4, c= 10^{-12}$, the roots approach 0 and $\frac{-b}{a}$.This case highlights the importance of floating-point precision.
4. **Error Handling**:
   - If the user inputs $a = 0$, the program terminates since the equation would not be quadratic.
5. **Possible Enhancements**:
   - **Graphical Representation**: Plot the quadratic function and mark its roots.
   - **Symbolic Computation**: Integrate with tools like **SymPy** to provide symbolic roots for exact values.
   - **User Input Validation**: Add error handling for invalid or non-numeric inputs.

Comparison and Related Concepts:

1. **Mathematical Background**:
   - Both programs involve mathematical computations—one for **statistics** (averages, deviations) and the other for **quadratic equations** (roots, discriminants).
2. **Applications**:
   - `score_sheet.c`: Useful in academic settings for analyzing student performance and generating reports.
   - `root.c`: Demonstrates how **polynomial equations** are solved, which has applications in physics, engineering, and data science.
3. **Software Design Patterns**:
   - Both programs use **modular design** with **functions** to separate logic, enhancing code readability and maintainability.

Conclusion:

The two programs demonstrate essential **programming concepts** like loops, conditional statements, mathematical calculations, and user input handling. Each program addresses a specific task, with room for further enhancements, such as **graphical visualization** and **data storage** for real-world use cases.
Both programs could benefit from further validation mechanisms and graphical tools to improve usability. However, they currently meet the requirements effectively, offering insight into **mathematical problem-solving** and **student performance evaluation**.