Code Report

Github Repository:

https://github.com/TheKurusUGM/Praktikum-Pemograman-UGM-/tree/708799223baed640f7b18f6cc02ce2c03eb7959c/Week_2

Problem 1. Payslip

Introduction:

The task was to create a simple C++ program that generates a payslip for an employee. The program should allow the user to input the employee's name, gross salary, installment amount, and insurance amount. The program then calculates the tax (20% of the gross salary), subtracts the tax, installment, and insurance amounts, and displays the net salary. The final output should present these values in a readable format with proper formatting, including thousands separators.

Final code:

```
#include <iostream>

#include <iomanip>

#include <locale>


int main() {


    std::string name;

    double grossSalary, netSalary;

    double tax, installment, insurance;



    const double TAX_RATE = 0.20;



    std::cout << "Enter employee name: ";

    std::getline(std::cin, name);
```

```
    std::cout << "Enter gross salary : ";
    std::cin >> grossSalary;

    std::cout << "Enter installment amount : ";
    std::cin >> installment;

    std::cout << "Enter insurance amount : ";
    std::cin >> insurance;


    tax = grossSalary * TAX_RATE;
    netSalary = grossSalary - tax - installment - insurance;


    std::cout.imbue(std::locale("en_US.UTF-8"));
    std::cout << std::fixed << std::setprecision(0);

    std::cout << "\n--- Employee Payslip ---\n";
    std::cout << "Name:          " << name << '\n';
    std::cout << "Gross Salary:  Rp" << grossSalary << '\n';
    std::cout << "Tax (20%):     Rp" << tax << '\n';
    std::cout << "Installment:   Rp" << installment << '\n';
    std::cout << "Insurance:     Rp" << insurance << '\n';
    std::cout << "Net Salary:    Rp" << netSalary << '\n';
    std::cout << "------------------------\n";

    return 0;
}
```

Final code runs as expected, it runs and fulfill all the required output, code uses <iomanip> and <locale> in order to give the output a better visual and more accurate. However, during the writing of the code, there are a couple of bugs and errors that have been fixed, here are some of the flaws found during the process of writing the code.

First error:

```
std::cout << "Enter employee name: ";
std::getline(std::cin, name);

std::cout << "Enter gross salary (in Rp): ";
std::cin >> grossSalary;

std::cout << "Enter installment amount (in Rp): ";
std::cin >> installment;

std::cout << "Enter insurance amount (in Rp): ";
std::cin >> insurance;
```

The initial code implementation successfully captured the employee's name and gross salary, but when the installment and insurance amounts were added, the program did not behave as expected. The specific issue was that the input for installment and insurance did not work correctly. To address the input issues, a revision to the code was written. The revision involved clearing the input buffer using std::cin.ignore() to avoid leftover newline characters interfering with subsequent inputs.

```
int main() {

    // Variables for employee details

    std::string name;

    double grossSalary, netSalary;

    double tax, installment, insurance;


    const double TAX_RATE = 0.20;  // 20% tax rate


    std::cout << "Enter employee name: ";

    std::getline(std::cin, name);


    std::cout << "Enter gross salary (in Rp): ";

    std::cin >> grossSalary;


    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

```
    std::cout << "Enter installment amount (in Rp): ";
    std::cin >> installment;

    std::cout << "Enter insurance amount (in Rp): ";
    std::cin >> insurance;

    tax = grossSalary * TAX_RATE;
    netSalary = grossSalary - tax - installment - insurance;

    std::cout << "\n--- Employee Payslip ---\n";
    std::cout << "Name:          " << name << '\n';
    std::cout << "Gross Salary:  Rp" << grossSalary << '\n';
    std::cout << "Tax (20%):     Rp" << tax << '\n';
    std::cout << "Installment:   Rp" << installment << '\n';
    std::cout << "Insurance:     Rp" << insurance << '\n';
    std::cout << "Net Salary:    Rp" << netSalary << '\n';
    std::cout << "------------------------\n";

    return 0;
}
```

However, the output of the program produced numerous errors and did not work as intended. This indicated that the proposed solution was not effective and may have introduced more complexity rather than resolving the original issue. The program first uses std::getline(std::cin, name); to read the employee's name, which can include spaces. For the rest of the inputs (grossSalary, installment, and insurance), then the code use std::cin >> to read the numbers directly. Since the program reads grossSalary, installment, and insurance using std::cin >>, there is no need for additional handling of the input buffer.

This however was the next output after the previous revision:

```
--- Employee Payslip ---
Name:          Pak Azhari
Gross Salary:  Rp2.5e+07
```

Tax (20%): Rp5e+06

Installment: Rp100000

Insurance: Rp1.2e+06

Net Salary: Rp1.87e+07

The output result such as Rp2.5e+07 instead of Rp25000000, is due to how floating-point numbers are formatted by default in C++. The numbers are displayed in scientific notation (e.g., 2.5e+07 means 2.5 * 10^7, which is 25000000).

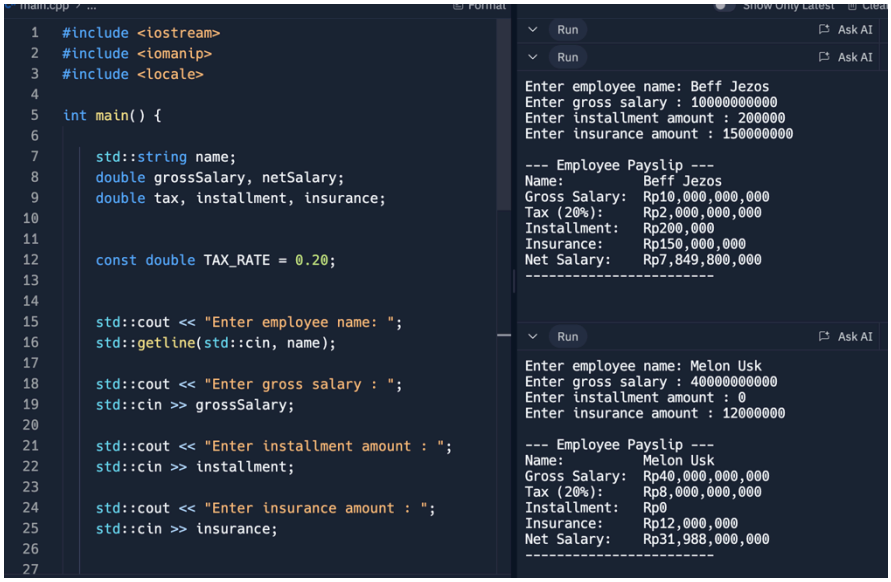
To fix this and ensure that the numbers are displayed in a regular decimal format, the code now use the std::fixed and std::setprecision manipulators from the <iomanip> library.

```
#include <iostream>

#include <iomanip> // Include for std::fixed and std::setprecision


int main() {

    // Variables for employee details

    std::string name;

    double grossSalary, netSalary;

    double tax, installment, insurance;

    const double TAX_RATE = 0.20;  // 20% tax rate


    std::cout << "Enter employee name: ";

    std::getline(std::cin, name);


    std::cout << "Enter gross salary (in Rp): ";

    std::cin >> grossSalary;


    std::cout << "Enter installment amount (in Rp): ";

    std::cin >> installment;


    std::cout << "Enter insurance amount (in Rp): ";
```

```
    std::cin >> insurance;
    tax = grossSalary * TAX_RATE;
    netSalary = grossSalary - tax - installment - insurance;

    std::cout << std::fixed << std::setprecision(0); // Force fixed-point with no decimal places
    std::cout << "\n--- Employee Payslip ---\n";
    std::cout << "Name:          " << name << '\n';
    std::cout << "Gross Salary:  Rp" << grossSalary << '\n';
    std::cout << "Tax (20%):     Rp" << tax << '\n';
    std::cout << "Installment:   Rp" << installment << '\n';
    std::cout << "Insurance:     Rp" << insurance << '\n';
    std::cout << "Net Salary:    Rp" << netSalary << '\n';
    std::cout << "------------------------\n";

    return 0;
}
```

The final improvement focused on enhancing the readability of the output by introducing commas as thousands separators. This was achieved by: Including the <locale> header. Ensuring that the output would display numbers with commas as thousands separators (e.g., 12,000,000 instead of 12000000). The final output met the requirements by correctly formatting the numbers in a readable manner with commas as thousands separators.

Test run:



```cpp
#include <iostream>
#include <iomanip>
#include <locale>

int main() {

    std::string name;
    double grossSalary, netSalary;
    double tax, installment, insurance;


    const double TAX_RATE = 0.20;


    std::cout << "Enter employee name: ";
    std::getline(std::cin, name);

    std::cout << "Enter gross salary : ";
    std::cin >> grossSalary;

    std::cout << "Enter installment amount : ";
    std::cin >> installment;

    std::cout << "Enter insurance amount : ";
    std::cin >> insurance;

```

```
Run  Ask AI
Run  Ask AI
Enter employee name: Beff Jezos
Enter gross salary : 10000000000
Enter installment amount : 200000
Enter insurance amount : 150000000

--- Employee Payslip ---
Name:           Beff Jezos
Gross Salary:   Rp10,000,000,000
Tax (20%):      Rp2,000,000,000
Installment:    Rp200,000
Insurance:      Rp150,000,000
Net Salary:     Rp7,849,800,000
------------------------
Run  Ask AI
Enter employee name: Melon Usk
Enter gross salary : 40000000000
Enter installment amount : 0
Enter insurance amount : 12000000

--- Employee Payslip ---
Name:           Melon Usk
Gross Salary:   Rp40,000,000,000
Tax (20%):      Rp8,000,000,000
Installment:    Rp0
Insurance:      Rp12,000,000
Net Salary:     Rp31,988,000,000
------------------------
```

Problem 2. Quadratic Equation solver

This report provides an overview of the quadratic equation solver program, its functionality, encountered bugs, and an analysis of test cases used to ensure its accuracy. The quadratic equation solver is designed to compute the roots of a quadratic equation of the form ax^2 + bx + c = 0 using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The program calculates the discriminant \Delta = b^2 - 4ac to determine the nature of the roots (real or complex) and outputs them accordingly.

The program prompts the user to input the coefficients a , b , and c for the quadratic equation. It then calculates the discriminant and uses it to determine the roots:

- $If\ \Delta > 0$: The program calculates two distinct roots

- $If\ \Delta = 0$: The program calculates one real root
- $If\ \Delta < 0$: The program calculates two complex roots

Final code:

```
#include <iostream>
#include <cmath>

int main() {
    double a, b, c;
    double discriminant, root1, root2;

    std::cout << "Enter coefficient a: ";
    std::cin >> a;
    std::cout << "Enter coefficient b: ";
    std::cin >> b;
    std::cout << "Enter coefficient c: ";
    std::cin >> c;

    if (a == 0) {
        std::cout << "This is not a quadratic equation (a cannot be 0)." << std::endl;
        return 1;
    }

    discriminant = b * b - 4 * a * c;

    if (discriminant > 0) {
        root1 = (-b + sqrt(discriminant)) / (2 * a);
        root2 = (-b - sqrt(discriminant)) / (2 * a);
        std::cout << "The equation has two distinct real roots:\n";
        std::cout << "Root 1: " << root1 << "\n";
```

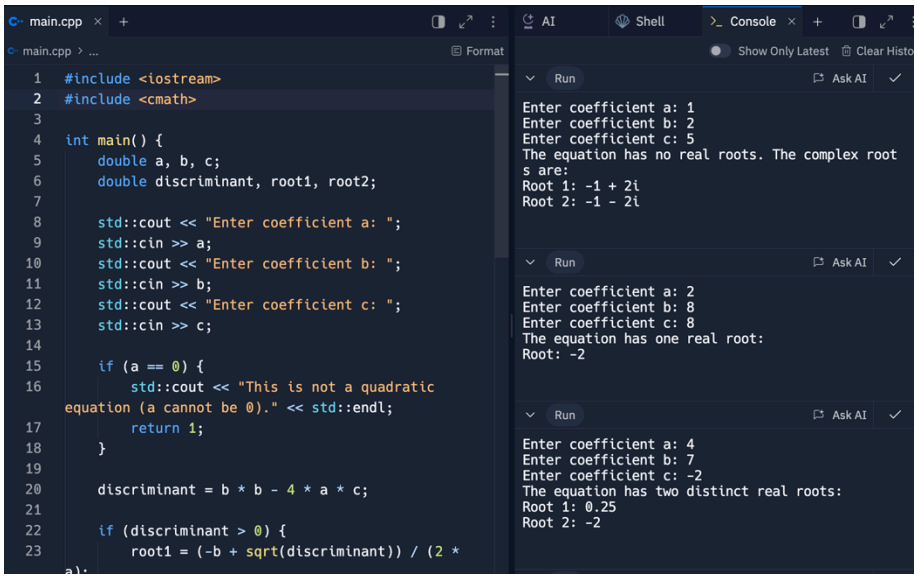
```
        std::cout << "Root 2: " << root2 << "\n";
    } else if (discriminant == 0) {
        root1 = -b / (2 * a);
        std::cout << "The equation has one real root:\n";
        std::cout << "Root: " << root1 << "\n";
    } else {
        double realPart = -b / (2 * a);
        double imaginaryPart = sqrt(-discriminant) / (2 * a);
        std::cout << "The equation has no real roots. The complex roots are:\n";
        std::cout << "Root 1: " << realPart << " + " << imaginaryPart << "i\n";
        std::cout << "Root 2: " << realPart << " - " << imaginaryPart << "i\n";
    }

    return 0;
}
```

During the development and testing of this program, several issues were identified:

Complex Roots Output:

- Description: When the discriminant was negative (\Delta < 0), the program correctly identified that the roots were complex. However, the output formatting of the complex roots was initially confusing due to the absence of proper formatting.
- Solution: The complex root output was improved by clearly separating the real and imaginary parts and explicitly showing the imaginary unit i .

Test run:


main.cpp
main.cpp > ...
Format
#include <iostream>
#include <cmath>
int main() {
double a, b, c;
double discriminant, root1, root2;
std::cout << "Enter coefficient a: ";
std::cin >> a;
std::cout << "Enter coefficient b: ";
std::cin >> b;
std::cout << "Enter coefficient c: ";
std::cin >> c;
if (a == 0) {
std::cout << "This is not a quadratic equation (a cannot be 0)." << std::endl;
return 1;
}
discriminant = b * b - 4 * a * c;
if (discriminant > 0) {
root1 = (-b + sqrt(discriminant)) / (2 * a);
AI
Shell
Console
Show Only Latest
Clear Histo
Run
Ask AI
Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 5
The equation has no real roots. The complex roots are:
Root 1: -1 + 2i
Root 2: -1 - 2i
Run
Ask AI
Enter coefficient a: 2
Enter coefficient b: 8
Enter coefficient c: 8
The equation has one real root:
Root: -2
Run
Ask AI
Enter coefficient a: 4
Enter coefficient b: 7
Enter coefficient c: -2
The equation has two distinct real roots:
Root 1: 0.25
Root 2: -2