Lab 07: CS 110

Function Arguments/Parameters, Variable Scope,

Debugging Basics

Muhammad Mujtaba

CMD ID: 540040

mmujtaba.bese25seecs@seecs.edu.pk

Class: BESE 16B

Batch: 2k25

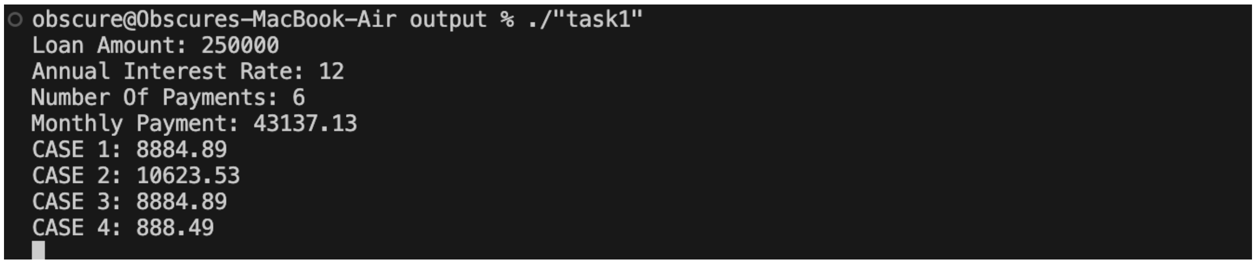
Task 1 [CLO 2]: Bank Loan Payment Calculator [Pass by Value + Default Parameters]

CODE:

```cpp
#include <iostream>
#include <cmath>
#include <string>
#include <iomanip>
#include <limits>

struct Range
{
    float min;
    float max;

    // get min and max of data type
    static Range getDefault()
    {
        return Range{std::numeric_limits<float>::min(), std::numeric_limits<float>::max()};
    }
};
// FUNCTIONS PROTOTYPES
float monthly_payment(float loan_amount, float annual_interest_rate, int number_of_payments =
12);
// struct is used so user needs to either define both min and max or none at all; to reduce
clutter
float input(const std::string &message, Range range = Range::getDefault());

// MAIN

int main()
{
    float loan_amount = input("Loan Amount", {0.0f, 1000000.0f});
    float annual_interest_rate = input("Annual Interest Rate", {0.0f, 100.0f});
    float number_of_payments = input("Number Of Payments", {1.0f, 600.0f}); // up to 50 years

    std::cout << std::fixed << std::setprecision(2);
    std::cout << "Monthly Payment: "
              << monthly_payment(loan_amount, annual_interest_rate, number_of_payments)
              << '\n';

    // test with default test cases
    std::cout << "CASE 1: " << monthly_payment(100000, 12, 12) << '\n';
    std::cout << "CASE 2: " << monthly_payment(500000, 10, 60) << '\n';
    std::cout << "CASE 3: " << monthly_payment(100000, 12) << '\n';
    std::cout << "CASE 4: " << monthly_payment(10000, 12) << '\n';

    std::cin.ignore();
    std::cin.get();
    return 0;
}
// FUNCTIONS DECLERATIONS
float monthly_payment(float loan_amount, float annual_interest_rate, int number_of_payments)
{
    // calculate monthly rate
    const float monthly_rate = annual_interest_rate / (12 * 100);
    // substituiting into formula
    return (monthly_rate * loan_amount) / (1 - std::pow(1 + monthly_rate, -
number_of_payments));
}
```

```cpp
float input(const std::string &message, Range range)
{
    float result = -1;
    std::cout << message << ": ";

    while (true)
    {
        std::cin >> result;
        // stopping the program from going crazy when alphabet is entered
        if (!std::cin.fail()) // if no error is detected in cin
        {
            if (result >= range.min && result <= range.max)
                break;

            std::cout << std::fixed << std::setprecision(2);
            std::cout << "> Input a number between " << range.min << " and " << range.max <<
": ";
            std::cout.unsetf(std::ios::fixed); // revert std::fixed
            continue;
        }

        // error in user input
        std::cin.clear();                                                    // clear error
flag
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // discard the
input
        // ^ says discard all input including newline character

        std::cout << "> Please input a valid input: ";
    }

    return result;
}
```

OUTPUT:

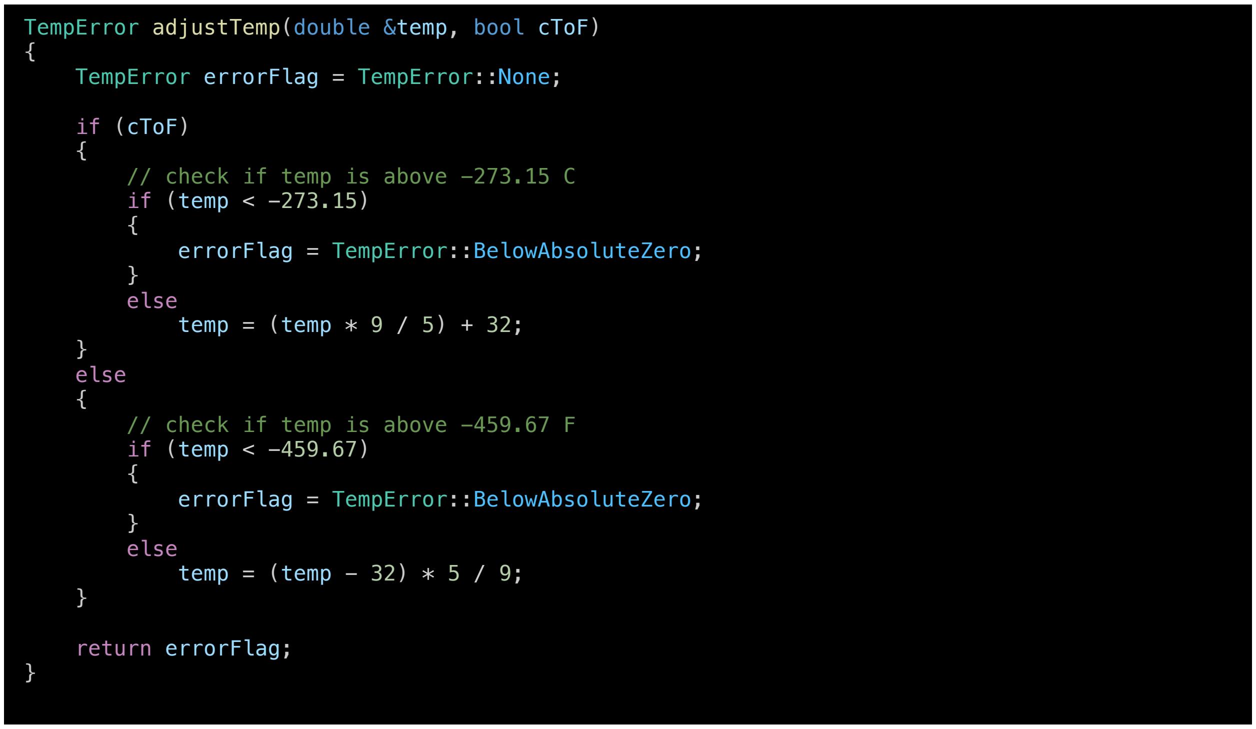


```
obscure@Obscures-MacBook-Air output % ./"task1"
Loan Amount: 250000
Annual Interest Rate: 12
Number Of Payments: 6
Monthly Payment: 43137.13
CASE 1: 8884.89
CASE 2: 10623.53
CASE 3: 8884.89
CASE 4: 888.49
```

Task 2 [CLO 2]: Celsius-Fahrenheit Converter [Pass by Reference]

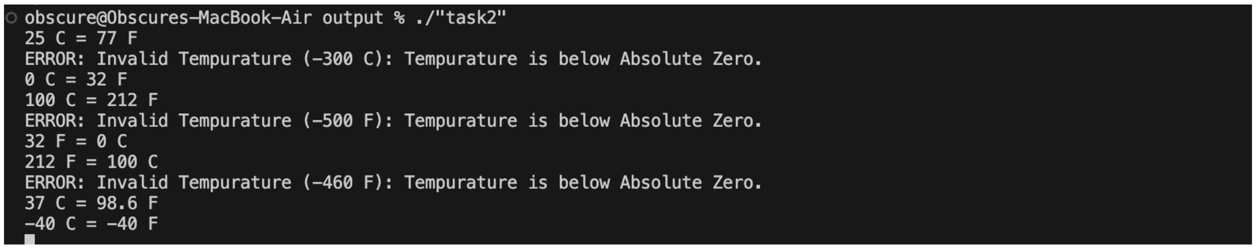
CODE:

```
#include <iostream>
#include <tuple>

enum class TempError
{
    None = 0,
    BelowAbsoluteZero = 1
};

TempError adjustTemp(double &temp, bool cToF);

int main()
{
    std::tuple<double, bool> testcases[10] = {
        {25.0, true},    // valid C
        {-300.0, true},  // invalid C
        {0.0, true},     // valid C
        {100.0, true},   // valid C
        {-500.0, false}, // invalid F
        {32.0, false},   // valid F
        {212.0, false},  // valid F
        {-460.0, false}, // invalid F
        {37.0, true},    // valid C
        {-40.0, true}    // same C and F
    };

    for (auto &[temp, isC] : testcases)
    {
        double tempBefore = temp;
        TempError error = adjustTemp(temp, isC);

        switch (error)
        {

        case TempError::None:
            std::cout << tempBefore << (isC ? " C" : " F") << " = " << temp << (!isC ? " C" :
" F") << "\n";
            break;

        case TempError::BelowAbsoluteZero:
            std::cout << "ERROR: Invalid Tempurature (" << tempBefore << "" << (isC ? " C" : "
F") << "): Tempurature is below Absolute Zero.\n";
            break;

        default:
            std::cout << "ERROR: Unknown Error Occured\n";
            break;
        }
    }

    std::cin.ignore();
    std::cin.get();
    return 0;
}
```



```cpp
TempError adjustTemp(double &temp, bool cToF)
{
    TempError errorFlag = TempError::None;

    if (cToF)
    {
        // check if temp is above -273.15 C
        if (temp < -273.15)
        {
            errorFlag = TempError::BelowAbsoluteZero;
        }
        else
            temp = (temp * 9 / 5) + 32;
    }
    else
    {
        // check if temp is above -459.67 F
        if (temp < -459.67)
        {
            errorFlag = TempError::BelowAbsoluteZero;
        }
        else
            temp = (temp - 32) * 5 / 9;
    }

    return errorFlag;
}
```

OUTPUT:



```
obscure@Obscures-MacBook-Air output % ./"task2"
25 C = 77 F
ERROR: Invalid Tempurature (-300 C): Tempurature is below Absolute Zero.
0 C = 32 F
100 C = 212 F
ERROR: Invalid Tempurature (-500 F): Tempurature is below Absolute Zero.
32 F = 0 C
212 F = 100 C
ERROR: Invalid Tempurature (-460 F): Tempurature is below Absolute Zero.
37 C = 98.6 F
-40 C = -40 F
```

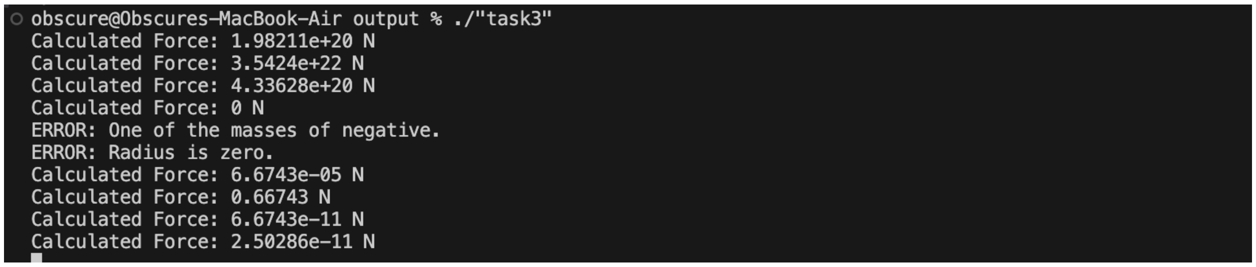
Task 3 [CLO 1]: Inline Functions

CODE:

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

enum class GravitationalForceError
{
    None = 0,
    ZeroRadius = -1,
    NegativeMass = -2,
};
inline double calculateForce(double first_mass, double second_mass, double radius,
GravitationalForceError *error);

int main()
{
    std::tuple<double, double, double> testcases[10] = {
        {5.972e24, 7.348e22, 3.844e8},    // Earth and Moon
        {1.989e30, 5.972e24, 1.496e11},   // Sun and Earth
        {1.989e30, 7.348e22, 1.496e11 + 3.844e8}, // Sun and Moon
        {5.972e24, 0.0, 3.844e8},         // Earth and zero mass
        {-5.972e24, 7.348e22, 3.844e8},   // Negative mass > ERROR
        {5.972e24, 7.348e22, 0.0},        // Zero radius > ERROR
        {1.0e3, 1.0e3, 1.0},              // Small masses
        {1.0e10, 1.0e10, 1.0e5},          // Large masses
        {1.0, 1.0, 1.0},                  // Unit masses
        {2.0, 3.0, 4.0}                   // Random values
    };

    for (const auto &[first_mass, second_mass, radius] : testcases)
    {
        GravitationalForceError error = GravitationalForceError::None;

        double result = calculateForce(first_mass, second_mass, radius, &error);

        switch (error)
        {

        case GravitationalForceError::None:
            std::cout << "Calculated Force: " << result << " N\n";
            break;

        case GravitationalForceError::ZeroRadius:
            std::cout << "ERROR: Radius is zero.\n";
            break;

        case GravitationalForceError::NegativeMass:
            std::cout << "ERROR: One of the masses of negative.\n";
            break;

        default:
            std::cout << "ERROR: Unknown Error Occured\n";
            break;
        }
    }

    std::cin.ignore();
    std::cin.get();
    return 0;
}
```

```
inline double calculateForce(double first_mass, double second_mass, double radius,
GravitationalForceError *error = nullptr)
{
    if (first_mass < 0 || second_mass < 0)
    {
        *error = GravitationalForceError::NegativeMass;
        return -1;
    }

    if (radius == 0)
    {
        *error = GravitationalForceError::ZeroRadius;
        return -2;
    }

    constexpr double G_CONSTANT = 6.67430e-11;

    return (G_CONSTANT * first_mass * second_mass) / (radius * radius);
}
```

OUTPUT:



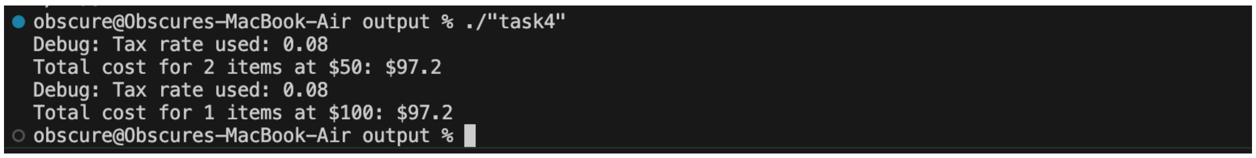
```
obscure@Obscures-MacBook-Air output % ./"task3"
Calculated Force: 1.98211e+20 N
Calculated Force: 3.5424e+22 N
Calculated Force: 4.33628e+20 N
Calculated Force: 0 N
ERROR: One of the masses of negative.
ERROR: Radius is zero.
Calculated Force: 6.6743e-05 N
Calculated Force: 0.66743 N
Calculated Force: 6.6743e-11 N
Calculated Force: 2.50286e-11 N
```

Task 4 [CLO 1]: Understanding scope of variables and debugging

CODE:

```
#include <iostream>

// Global variable for tax rate
double taxRate = 0.08; // 8% tax rate

double calculateFinalPrice(double price, int quantity)
{
    double total; // FIX 1: initialize
    double discount = 0.1;
    total = price * quantity * (1 + taxRate) * (1 - discount); // FIX 2: use correct formula
    // FIX 3: remove local `taxRate` variable
    // FIX 4: Remove unneccessay scope

    std::cout << "Debug: Tax rate used: " << taxRate << std::endl;
    return total;
}

int main()
{
    double price = 50.0;
    int quantity = 2;
    double expectedTotal = (price * quantity) * (1 + 0.08) * (1 - 0.1);
    // Expected: 50 * 2 * 1.08 * 0.9 = 97.2

    double result = calculateFinalPrice(price, quantity);
    std::cout << "Total cost for " << quantity << " items at $" << price
              << ": $" << result << std::endl;

    // Test with different values
    price = 100.0;
    quantity = 1;
    expectedTotal = (price * quantity) * (1 + 0.08) * (1 - 0.1); // Expected: 100 * 1 * 1.08 *
0.9 = 97.2
    result = calculateFinalPrice(price, quantity);
    std::cout << "Total cost for " << quantity << " items at $" << price
              << ": $" << result << std::endl;

    return 0;
}
```

OUTPUT:



```
● obscure@Obscures-MacBook-Air output % ./"task4"
  Debug: Tax rate used: 0.08
  Total cost for 2 items at $50: $97.2
  Debug: Tax rate used: 0.08
  Total cost for 1 items at $100: $97.2
○ obscure@Obscures-MacBook-Air output %
```