RBE104TC

Semester1 Course Work 2

author: Xi Nie

Student ID: 2255693

Date: Friday 22$^{\text{nd}}$ September, 2023

I. Software Development Process (SDP)

1.1 Problem Statement:

1. Exercise 1

- Part 1

· Create a class specifically for representing fractions.

· The fraction class requires two integer private member variables, top and bottom, for the numerator and denominator.

· The class must implement operator overloading to perform fraction addition, subtraction, multiplication, and division.

· To compare fractions, the class must overload relational operations.

· Concerns overloading input and output operators.

- Part 2

· Negative sign needed in the numerator.

· Find the greatest common divisor and divide the numerator denominator by it.

· Converting fractions and decimals requires two functions.

2. Exercise 2

- Part 1

· Create the subclass iFraction with an initialization constructor and a method to display the mixed fraction.

- Part 2

· Create a non-Fraction or iFraction external function convertF to convert improper fractions to mixed fractions.

3. Exercise3

· A composite class representing complex numbers with fractional real and imaginary portions should contain its constructor and overload complex number operations.

4. The program exists.

II. EXERCISE 1

2.1 Analysis for exercise 1:

• On an input:
Create an object of the Fraction class and pass a different number of parameters.

• On an output:
Output the simplest fraction, representing the numerator/denominator

• Data structure:
Only single data is involved.

• Algorithm:
The four operations with fractions, simplification of fractions

2.2 Design for exercise 1:

i. Build Fraction.h

Definitions and Overloading

Declare two private member variables and name them sensibly.

Creates three public constructors with different numbers of parameters.

Initialisation is performed outside the Fraction class.

Operator overloading for fractions.

· Add and subtract the fractions by cross-multiplying.

· Divide and multiply the numerators and denominators.

Overloading six relational operators

· Overloading of Relational Operators by cross-multiplying.

Define two friend functions to overload the input and output operators.

• Input:

Defines a variable to save the input stream slash character.

Data is read and assigned to Fraction object member variables using the input stream object extraction operator.

The is stream object reads and assigns input stream data to the fraction object's top member variable, then *the slash character, and lastly the bottom member variable.*

The function returns the input stream reference last.

• Output:

Identify the numerator and denominator as positive or negative.

If they match, print the numerator-denominator absolute value.

If not, output the opposite of the numerator denominator's absolute value.

The output stream and a const Fraction object reference are the first and second arguments, respectively.

Function returns are output stream references.

Define two public member functions findGCD() and simplify() to simplify the fractions.

ii. Build main function

Declare variables, assign values, and print out the results.

iii. Build Transformer.cpp

- decimalToFraction function.

This method returns a Fraction object from a double-value decimal.

It initializes variables such as maxIterations (maximum number of iterations), epsilon (tolerance for approximation), sign (sign of the decimal number), wholePart (integer part of the decimal number), fractionPart (fractional part of the decimal number), numerator (numerator of the fraction), and denominator (denominator of the fraction).

The loop calculates the decimal number's continuing fraction until accuracy is attained or the maximum number of iterations is reached

In each iteration, the loop updates the fractionPart, numerator, wholePart, and denominator.

The function produces a Fraction object result using the computed values following fraction simplification.

- fractionToDecimal function

This function converts a Fraction object fraction to a double

The function retrieves the fraction's decimal value from the Fraction class's toDecimal method.

III. EXERCISE 2

3.1 Analysis for exercise 2:

- On an input: Fraction

- On an output: mixed fractions
- Data structure:

Only one data is involved.

- Algorithm:

The equations may be easily evaluated. There are no discernible issues or challenges associated with algorithms.

3.2 Design for exercise 2:

i. create a iFraction() class base on Fraction()

Create a constructor and call the parent class Fraction constructor with sTop and sBottom.

Create the iFraction class member method MixedFraction() to compute and report mixed fractions.

Multiply sTop and sBottom to check the positive and negative and compute the integer component and numerator.

Print "mixed fractions: integer numerator/denominator" with cout.

ii. The parent and child classes should also have the friend method convertF with fraction arguments.

iii. advanced main function

Creating objects with iFraction and assigning values for printing.

The convertF function is called, passed parameters, and returns the mixed fraction.

IV. EXERCISE 3

4.1 Analysis for exercise 3:

• On an input: Fraction type of fraction, Complex numbers, Operators

• On an output: Complex number

• Data structure:

There is just a single datum involved. There are no challenges related to sophisticated data structures in this context.

• Algorithm:

Evaluation of the equations is uncomplicated. Similarly, there are no issues with algorithms.

4.2 Design for exercise 3:

i. Build Complex.cpp

Private Part

Declare two private Fraction type members, realPart and imaginaryPart, to represent the complex number's real and imaginary parts.

Public Part

Multiply sTop and sBottom to check the positive and negative and compute the integer component and numerator.

Print "mixed fractions: integer numerator/denominator" with cout.

ii. The parent and child classes should also have the friend method convertF with fraction arguments.

Operator overloading for complex.

· *Addition and subtraction add or remove real and imaginary elements.*

· *Complex multiplication and division formulae are used.*

Define void print() to print complex numbers' real and imaginary portions.

iii. advanced main function

Three Fraction objects are created, and two Complex objects are created from the Fraction objects.

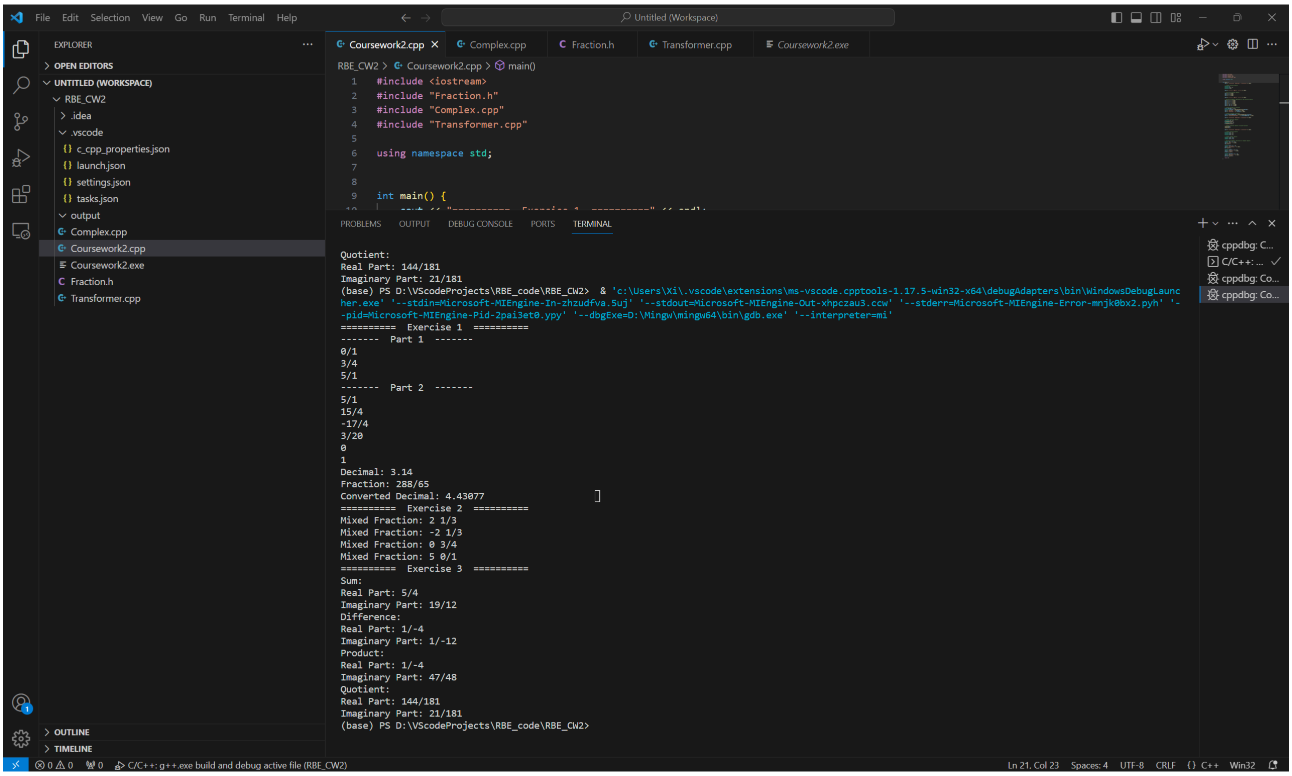
Use the overloaded operators, the calculation.

The print function is called to print it out.

V. Implementation

Please see the C++ source code (at the end of this document) with comments.

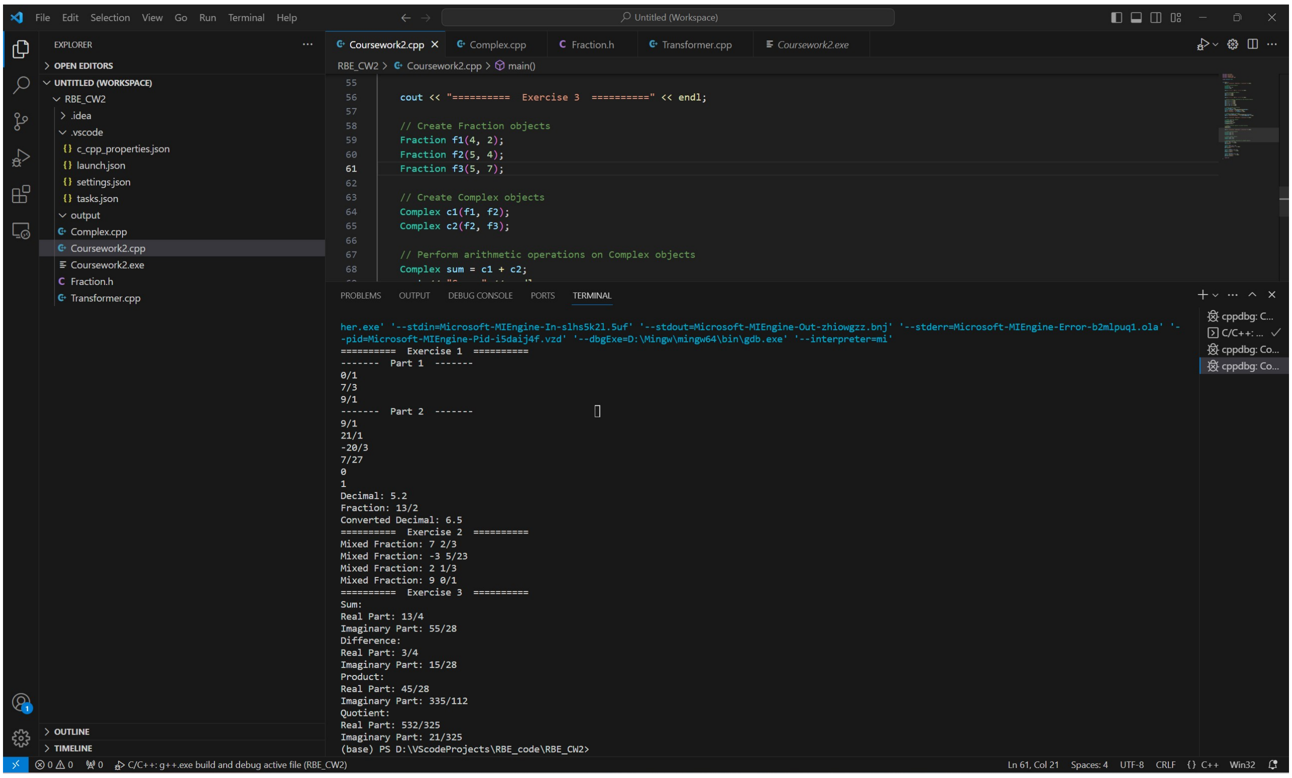
VI. Testing:



```cpp
#include <iostream>
#include "Fraction.h"
#include "Complex.cpp"
#include "Transformer.cpp"

using namespace std;


int main() {
```

```
Quotient:
Real Part: 144/181
Imaginary Part: 21/181
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>  & 'c:\Users\Xi\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLaunc
her.exe' '--stdin=Microsoft-MIEngine-In-zhzudfva.5uj' '--stdout=Microsoft-MIEngine-Out-xhpczau3.ccw' '--stderr=Microsoft-MIEngine-Error-mnjk0bx2.pyh' '-
-pid=Microsoft-MIEngine-Pid-2pai3et0.ypy' '--dbgExe=D:\Mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
==========  Exercise 1  ==========
-------  Part 1  -------
0/1
3/4
5/1
-------  Part 2  -------
5/1
15/4
-17/4
3/20
0
1
Decimal: 3.14
Fraction: 288/65
Converted Decimal: 4.43077
==========  Exercise 2  ==========
Mixed Fraction: 2 1/3
Mixed Fraction: -2 1/3
Mixed Fraction: 0 3/4
Mixed Fraction: 5 0/1
==========  Exercise 3  ==========
Sum:
Real Part: 5/4
Imaginary Part: 19/12
Difference:
Real Part: 1/-4
Imaginary Part: 1/-12
Product:
Real Part: 1/-4
Imaginary Part: 47/48
Quotient:
Real Part: 144/181
Imaginary Part: 21/181
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>
```

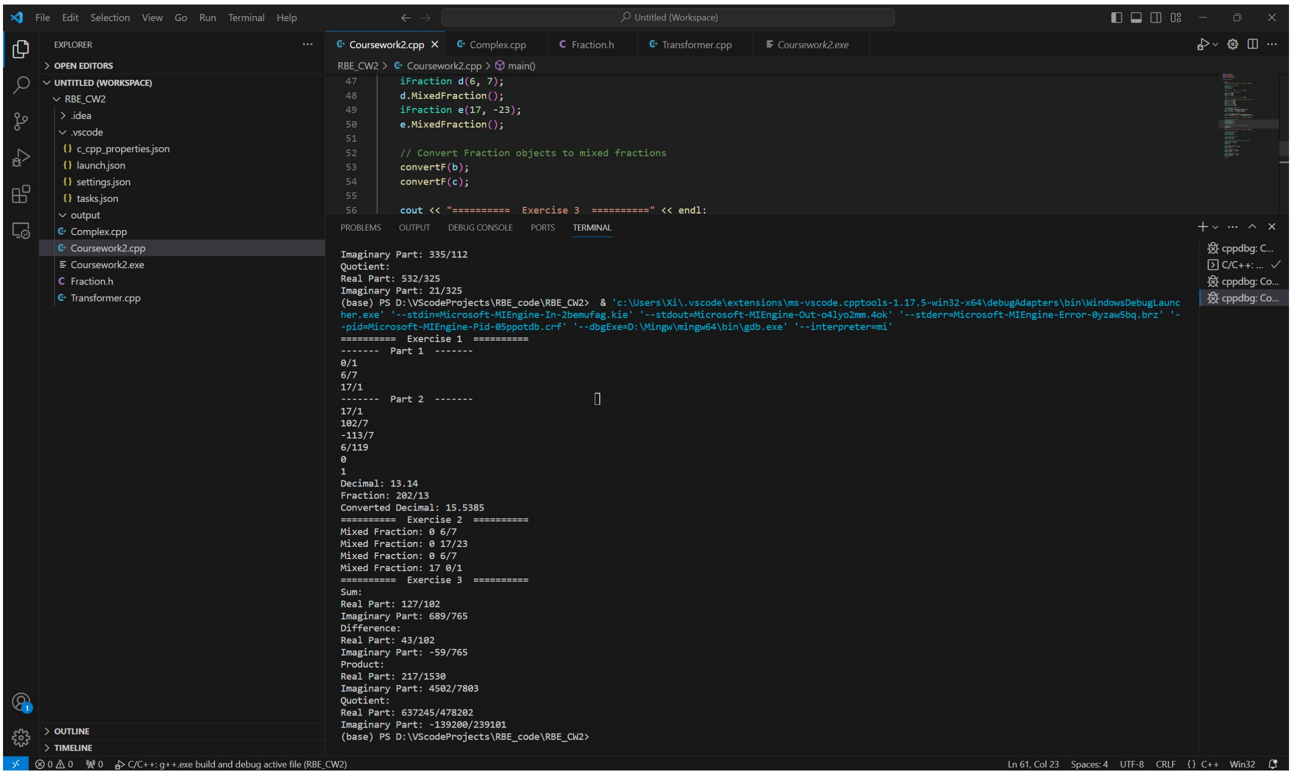
Figure 1: Test1



```cpp

    cout << "==========  Exercise 3  ==========" << endl;

    // Create Fraction objects
    Fraction f1(4, 2);
    Fraction f2(5, 4);
    Fraction f3(5, 7);

    // Create Complex objects
    Complex c1(f1, f2);
    Complex c2(f2, f3);

    // Perform arithmetic operations on Complex objects
    Complex sum = c1 + c2;
```

```
her.exe' '--stdin=Microsoft-MIEngine-In-slhs5k2l.5uf' '--stdout=Microsoft-MIEngine-Out-zhiowgzz.bnj' '--stderr=Microsoft-MIEngine-Error-b2mlpuq1.ola' '-
-pid=Microsoft-MIEngine-Pid-i5daij4f.vzd' '--dbgExe=D:\Mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
==========  Exercise 1  ==========
-------  Part 1  -------
0/1
7/3
9/1
-------  Part 2  -------
9/1
21/1
-20/3
7/27
0
1
Decimal: 5.2
Fraction: 13/2
Converted Decimal: 6.5
==========  Exercise 2  ==========
Mixed Fraction: 7 2/3
Mixed Fraction: -3 5/23
Mixed Fraction: 2 1/3
Mixed Fraction: 9 0/1
==========  Exercise 3  ==========
Sum:
Real Part: 13/4
Imaginary Part: 55/28
Difference:
Real Part: 3/4
Imaginary Part: 15/28
Product:
Real Part: 45/28
Imaginary Part: 335/112
Quotient:
Real Part: 532/325
Imaginary Part: 21/325
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>
```

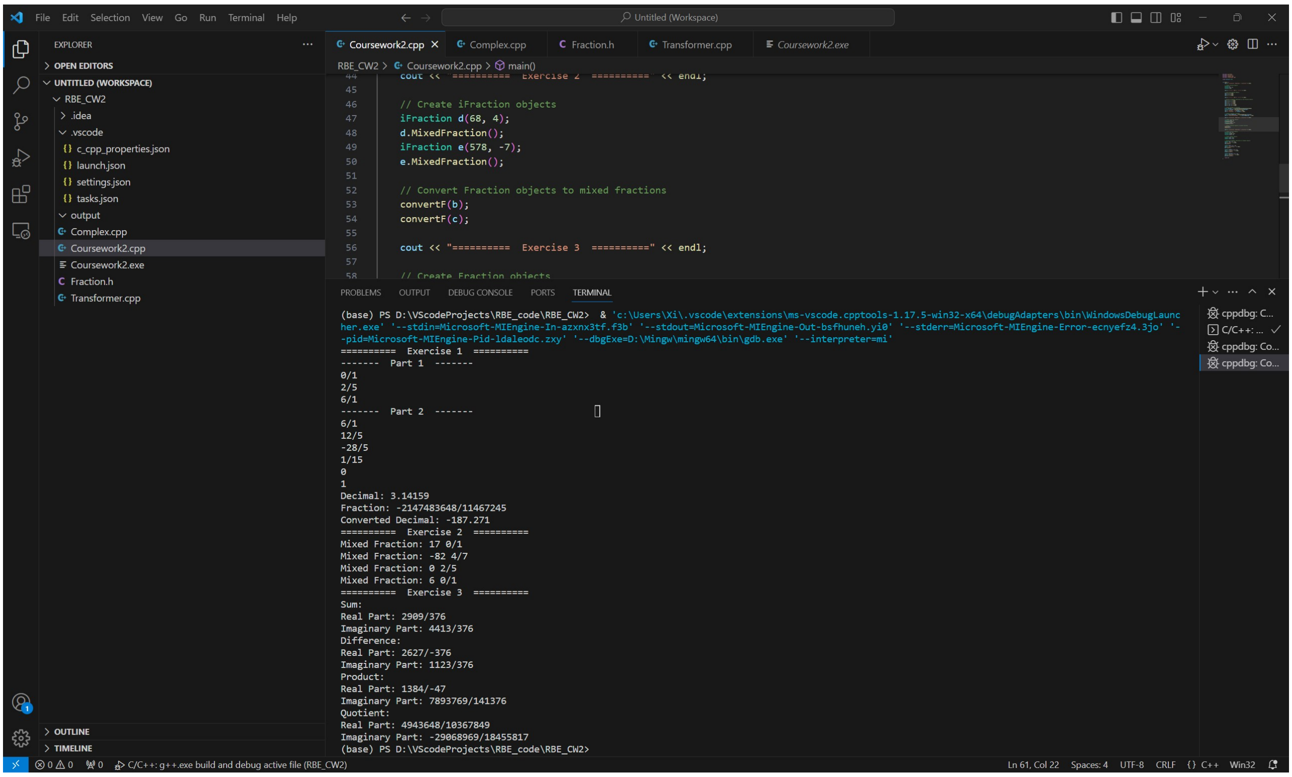
Figure 2: Test2



```cpp
    iFraction d(6, 7);
    d.MixedFraction();
    iFraction e(17, -23);
    e.MixedFraction();

    // Convert Fraction objects to mixed fractions
    convertF(b);
    convertF(c);

    cout << "==========  Exercise 3  ==========" << endl;
```

```
Imaginary Part: 335/112
Quotient:
Real Part: 532/325
Imaginary Part: 21/325
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>  & 'c:\Users\Xi\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-2bemufag.kie' '--stdout=Microsoft-MIEngine-Out-o4lyo2mm.4ok' '--stderr=Microsoft-MIEngine-Error-0yzaw5bq.brz' '--pid=Microsoft-MIEngine-Pid-05ppotdb.crf' '--dbgExe=D:\Mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
==========  Exercise 1  ==========
-------  Part 1  -------
0/1
6/7
17/1
-------  Part 2  -------
17/1
102/7
-113/7
6/119
0
1
Decimal: 13.14
Fraction: 202/13
Converted Decimal: 15.5385
==========  Exercise 2  ==========
Mixed Fraction: 0 6/7
Mixed Fraction: 0 17/23
Mixed Fraction: 0 6/7
Mixed Fraction: 17 0/1
==========  Exercise 3  ==========
Sum:
Real Part: 127/102
Imaginary Part: 689/765
Difference:
Real Part: 43/102
Imaginary Part: -59/765
Product:
Real Part: 217/1530
Imaginary Part: 4502/7803
Quotient:
Real Part: 637245/478202
Imaginary Part: -139200/239101
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>
```

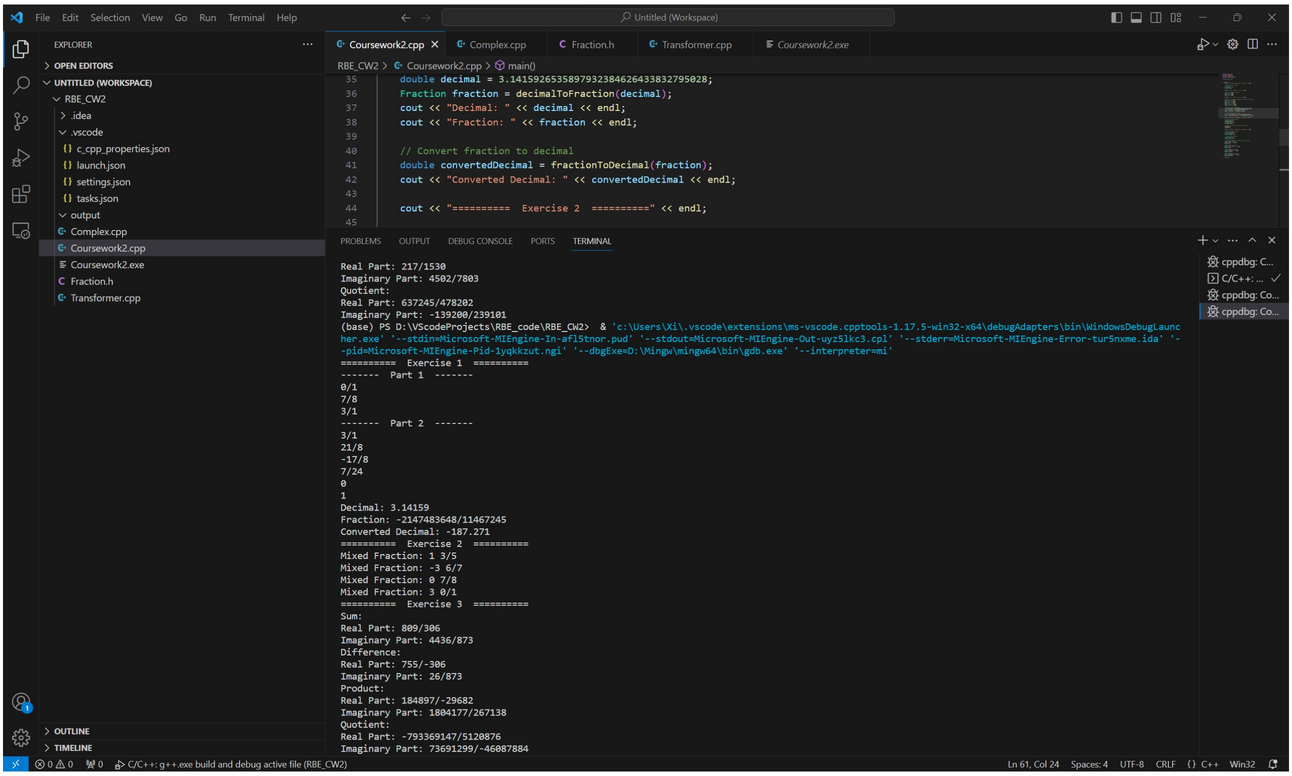
Figure 3: Test3



```cpp
    cout << "==========  Exercise 2  ==========" << endl;

    // Create iFraction objects
    iFraction d(68, 4);
    d.MixedFraction();
    iFraction e(578, -7);
    e.MixedFraction();

    // Convert Fraction objects to mixed fractions
    convertF(b);
    convertF(c);

    cout << "==========  Exercise 3  ==========" << endl;

    // Create Fraction objects
```

```
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>  & 'c:\Users\Xi\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-azxnx3tf.f3b' '--stdout=Microsoft-MIEngine-Out-bsfhuneh.yi0' '--stderr=Microsoft-MIEngine-Error-ecnyefz4.3jo' '--pid=Microsoft-MIEngine-Pid-ldaleodc.zxy' '--dbgExe=D:\Mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
==========  Exercise 1  ==========
-------  Part 1  -------
0/1
2/5
6/1
-------  Part 2  -------
6/1
12/5
-28/5
1/15
0
1
Decimal: 3.14159
Fraction: -2147483648/11467245
Converted Decimal: -187.271
==========  Exercise 2  ==========
Mixed Fraction: 17 0/1
Mixed Fraction: -82 4/7
Mixed Fraction: 0 2/5
Mixed Fraction: 6 0/1
==========  Exercise 3  ==========
Sum:
Real Part: 2909/376
Imaginary Part: 4413/376
Difference:
Real Part: 2627/-376
Imaginary Part: 1123/376
Product:
Real Part: 1384/-47
Imaginary Part: 7893769/141376
Quotient:
Real Part: 4943648/10367849
Imaginary Part: -29068969/18455817
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>
```

Figure 4: Test4



```cpp
    double decimal = 3.14159265358979323846264338327950288;
    Fraction fraction = decimalToFraction(decimal);
    cout << "Decimal: " << decimal << endl;
    cout << "Fraction: " << fraction << endl;

    // Convert fraction to decimal
    double convertedDecimal = fractionToDecimal(fraction);
    cout << "Converted Decimal: " << convertedDecimal << endl;

    cout << "==========  Exercise 2  ==========" << endl;

```

```
Real Part: 217/1530
Imaginary Part: 4502/7803
Quotient:
Real Part: 637245/478202
Imaginary Part: -139200/239101
(base) PS D:\VScodeProjects\RBE_code\RBE_CW2>  & 'c:\Users\Xi\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-afl5tnor.pud' '--stdout=Microsoft-MIEngine-Out-uyz5lkc3.cpl' '--stderr=Microsoft-MIEngine-Error-tur5nxme.ida' '--pid=Microsoft-MIEngine-Pid-1yqkkzut.ngi' '--dbgExe=D:\Mingw\mingw64\bin\gdb.exe' '--interpreter=mi'
==========  Exercise 1  ==========
-------  Part 1  -------
0/1
7/8
3/1
-------  Part 2  -------
3/1
21/8
-17/8
7/24
0
1
Decimal: 3.14159
Fraction: -2147483648/11467245
Converted Decimal: -187.271
==========  Exercise 2  ==========
Mixed Fraction: 1 3/5
Mixed Fraction: -3 6/7
Mixed Fraction: 0 7/8
Mixed Fraction: 3 0/1
==========  Exercise 3  ==========
Sum:
Real Part: 809/306
Imaginary Part: 4436/873
Difference:
Real Part: 755/-306
Imaginary Part: 26/873
Product:
Real Part: 184897/-29682
Imaginary Part: 1804177/267138
Quotient:
Real Part: -793369147/5120876
Imaginary Part: 73691299/-46087884
```

Figure 5: Test5

VII. C++ Code

7.1 Courseworks2.cpp

```cpp
{c++}
/*
Name: RBE104_Courseworks2
File Name: Courseworks2.cpp
Author: Xi Nie
Description:
this program showcases the functionality of the Fraction and Complex classes, providing a comprehensive understanding of working with fractions and complex numbers in C++.

*/
#include <iostream>
#include "Fraction.h"
#include "Complex.cpp"
#include "Transformer.cpp"

using namespace std;


int main() {
    cout << "========== Exercise 1 ==========" << endl;

    // Create Fraction objects
    Fraction a;
    Fraction b(3, 4);
    Fraction c(5);

    cout << "------- Part 1 -------" << endl;

    // Print the Fraction objects
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    cout << "------- Part 2 -------" << endl;

    // Perform arithmetic operations on the Fraction objects
    cout << a + c << endl;
    cout << b * c << endl;
    cout << b - c << endl;
    cout << b / c << endl;
    cout << (b > c) << endl;
    cout << (b < c) << endl;

    // Convert decimal to fraction
    double decimal = 3.14;
    Fraction fraction = decimalToFraction(decimal);
    cout << "Decimal: " << decimal << endl;
    cout << "Fraction: " << fraction << endl;

    // Convert fraction to decimal
    double convertedDecimal = fractionToDecimal(fraction);
    cout << "Converted Decimal: " << convertedDecimal << endl;

    cout << "========== Exercise 2 ==========" << endl;

    // Create iFraction objects
    iFraction d(7, 3);
    d.MixedFraction();
    iFraction e(-7, 3);
    e.MixedFraction();

    // Convert Fraction objects to mixed fractions
    convertF(b);
    convertF(c);

    cout << "========== Exercise 3 ==========" << endl;

    // Create Fraction objects
    Fraction f1(1, 2);
    Fraction f2(3, 4);
    Fraction f3(5, 6);

    // Create Complex objects
    Complex c1(f1, f2);
    Complex c2(f2, f3);

    // Perform arithmetic operations on Complex objects
    Complex sum = c1 + c2;
    cout << "Sum: " << endl;
    sum.print();

    Complex diff = c1 - c2;
    cout << "Difference: " << endl;
    diff.print();

    Complex product = c1 * c2;
    cout << "Product: " << endl;
    product.print();

    Complex quotient = c1 / c2;
    cout << "Quotient: " << endl;
    quotient.print();

    return 0;
}

```

Notes:

7.2 Fraction.h

```
{c++}
/*
Name: RBE104_Courseworks2
File Name: Fraction.h
Author: Xi Nie
Description:
This program offers functionality for working with and mixing fractions conveniently and efficiently.


*/
#ifndef FRACTION_H
#define FRACTION_H
#include<iostream>
using namespace std;


class Fraction {
private:
    int top;
    int bottom;



public:
    //constructor
    Fraction();
    Fraction(int num);
    Fraction(int num, int denom);

    /*Overloading of binomial arithmetic operators
        Add and subtract the fractions by cross-multiplying.
        Divide and multiply the numerators and denominators.
    */
    Fraction operator+(const Fraction& other) const {
        int num = (top * other.bottom) + (other.top * bottom);
        int denom = bottom * other.bottom;
        return Fraction(num, denom);
    }

    Fraction operator-(const Fraction& other) const {
        int num = (top * other.bottom) - (other.top * bottom);
        int denom = bottom * other.bottom;
        return Fraction(num, denom);
    }

    Fraction operator*(const Fraction& other) const {
        int num = top * other.top;
        int denom = bottom * other.bottom;
        return Fraction(num, denom);
    }

    Fraction operator/(const Fraction& other) const {
        int num = top * other.bottom;
        int denom = bottom * other.top;
        return Fraction(num, denom);
    }


    // Overloading of Relational Operators by cross-multiplying
    bool operator==(const Fraction& other) const {
        return (top * other.bottom) == (other.top * bottom);
    }

    bool operator!=(const Fraction& other) const {
        return !(*this == other);
    }

    bool operator<(const Fraction& other) const {
        return (top * other.bottom) < (other.top * bottom);
    }

    bool operator>(const Fraction& other) const {
        return (top * other.bottom) > (other.top * bottom);
    }

    bool operator<=(const Fraction& other) const {
        return (*this < other) || (*this == other);
    }

    bool operator>=(const Fraction& other) const {
        return (*this > other) || (*this == other);
    }

    /*
     Overloaded << and >> operators in Fraction class for buddy functions.
     Fraction objects are output and input by them.
     */
    friend ostream& operator<<(ostream& os, const Fraction& fraction) {
        if (fraction.top*fraction.bottom<0)
        {
            os << - abs(fraction.top) << "/" << abs(fraction.bottom);
        }else{
            os << abs(fraction.top) << "/" << abs(fraction.bottom);
        }

        return os;
    }

    friend istream& operator>>(istream& is, Fraction& fraction) {
        char slash;
        is >> fraction.top >> slash >> fraction.bottom;
        return is;
    }

    //Declare a friend function
    friend void convertF(Fraction& fraction);

    double toDecimal() const {
        return static_cast<double>(top) / bottom;
    }

    // Accessor method to get the private top and bottom value.
    int getPrivateTop() const {
        return top;
    }
    int getPrivateBottom() const {
        return bottom;
    }

    // Print the fraction in the form of "top/bottom"
    void print() const {
        cout << top << "/" << bottom << endl;
    };



    /*Calculate GCD using recursion.
```

```
        The function takes two fraction numerator and denominator parameters.
        The code yields a when b = 0.
        The recursive function calls itself with b as the numerator and the residue of a divided by b as the denominator.
        Repeat until the base case is reached and GCD is returned.
    */
    int findGCD(int a, int b) {
        // Base case: if b is 0, return a
        if (b == 0)
            return a;
        // Recursive case: compute the GCD using modulo operator
        return findGCD(b, a % b);
    }
    // Find the greatest common divisor using recursion
    void simplify() {
        int gcd = findGCD(top, bottom);
        top /= gcd;
        bottom /= gcd;
    }
};

/*Depending on the number of input data, the corresponding constructor is initialized.
    Default constructor: initialize top to 0 and bottom to 1.
    Constructor with one argument: initialize top to num and bottom to 1.
    Constructor with two arguments: initialize top to num and bottom to denom.
*/
Fraction::Fraction() {
    top = 0;
    bottom = 1;
}
Fraction::Fraction(int num) {
    top = num;
    bottom = 1;
}
Fraction::Fraction(int num, int denom) {
    top = num;
    bottom = denom;
    simplify();
}

// Convert the fraction to a mixed fraction
void convertF(Fraction& fraction) {
    int fWhole = fraction.top / fraction.bottom;
    int fRemainder = fraction.top % fraction.bottom;

    cout << "Mixed Fraction: " << fWhole << " " << fRemainder << "/" << fraction.bottom << endl;
}

//Define a derived class based on Fraction()
class iFraction: public Fraction{
public:
    //Constructor for iFraction
    iFraction(int sTop, int sBottom) : Fraction(sTop, sBottom) {}
    int sTop = getPrivateTop();
    int sBottom = getPrivateBottom();

    // Convert the fraction to a mixed fraction
    void MixedFraction (){
        int numerator,integer;
        if (sTop * sBottom<0)
        {
            integer = - abs(sTop) / abs(sBottom);
            numerator = abs(sTop) % abs(sBottom);
        }else{
            integer = sTop / sBottom;
            numerator = sTop % sBottom;
        }

        cout << "Mixed Fraction: " << integer << " " << numerator << "/" << abs(sBottom) << endl;

    };

    friend void convertF(iFraction& fraction);
};

void convertF(iFraction& fraction) {
    // Convert the fraction to a mixed fraction
    int sWhole = fraction.sTop / fraction.sBottom;
    int sRemainder = fraction.sTop % fraction.sBottom;

    cout << "Mixed Fraction: " << sWhole << " " << sRemainder << "/" << fraction.sBottom << endl;
}



#endif
```

Notes:

7.3 Transformer.cpp

```
{c++}
/*
Name: RBE104_Courseworks2
File Name: Transformer.cpp
Author: Xi Nie
Description:
This program offers functionality for converting between decimal numbers and fractions.
*/


#include "Fraction.h" // Include the Fraction header file
#include <iostream>
#include <cmath>

using namespace std;

// Function to convert a decimal number to a fraction
Fraction decimalToFraction(double decimal) {
    const int maxIterations = 10; // Maximum number of iterations for approximation
    double epsilon = 1.0e-6; // Tolerance for approximation

    int sign = 1; // Sign of the decimal number
    if (decimal < 0) {
        sign = -1;
        decimal = -decimal; // Convert decimal to positive for calculation
    }

    int wholePart = static_cast<int>(decimal); // Integer part of the decimal number

    double fractionPart = decimal - wholePart; // Fractional part of the decimal number
    double numerator = fractionPart; // Numerator of the fraction
    double denominator = 1; // Denominator of the fraction

    // Approximate the fraction part using continued fraction expansion
    for (int i = 2; i <= maxIterations && fabs(fractionPart - static_cast<int>(fractionPart)) > epsilon; i++) {
        fractionPart = 1 / (fractionPart - static_cast<int>(fractionPart)); // Calculate the next term of the continued fraction
        int tempNumerator = numerator;
        numerator = fractionPart * numerator + wholePart; // Update the numerator
        wholePart = tempNumerator; // Update the whole part
        denominator = fractionPart * denominator + denominator; // Update the denominator
    }

    Fraction result(sign * (wholePart * denominator + numerator), denominator); // Create a Fraction object with the calculated values
    result.simplify(); // Simplify the fraction

    return result; // Return the resulting fraction
}

// Function to convert a fraction to a decimal number
double fractionToDecimal(const Fraction& fraction) {
    return fraction.toDecimal(); // Call the toDecimal() method of the Fraction class to calculate the decimal value of the fraction
}


```

Notes:

7.4 Complex.cpp

```
{c++}
/*
Name: RBE104_Courseworks2
File Name: Complex.cpp
Author: Xi Nie
Description:
This program provides functionality for working with complex numbers and performing arithmetic operations.


*/
#include <iostream>
#include "Fraction.h"


class Complex {
private:
    Fraction realPart;
    Fraction imaginaryPart;

public:
    Complex(Fraction real, Fraction imaginary) {
        realPart = real;
        imaginaryPart = imaginary;
    }
    /*Overloading of binomial arithmetic operators
        Addition and subtraction add or remove real and imaginary elements.
        Complex multiplication and division formulae are used.
    */
    Complex operator+(const Complex& other) const {
        return Complex(realPart + other.realPart, imaginaryPart + other.imaginaryPart);
    }

    Complex operator-(const Complex& other) const {
        return Complex(realPart - other.realPart, imaginaryPart - other.imaginaryPart);
    }

    Complex operator*(const Complex& other) const {
        Fraction real = realPart * other.realPart - imaginaryPart * other.imaginaryPart;
        Fraction imaginary = realPart * other.imaginaryPart + imaginaryPart * other.realPart;
        return Complex(real, imaginary);
    }

    Complex operator/(const Complex& other) const {
        Fraction divisor = other.realPart * other.realPart + other.imaginaryPart * other.imaginaryPart;
        Fraction real = (realPart * other.realPart + imaginaryPart * other.imaginaryPart) / divisor;
        Fraction imaginary = (imaginaryPart * other.realPart - realPart * other.imaginaryPart) / divisor;
        return Complex(real, imaginary);
    }
    /*
       Prints the real and imaginary parts of the complex number
    */
    void print() const {
        cout << "Real Part: ";
        realPart.print();
        cout << "Imaginary Part: ";
        imaginaryPart.print();
    }
};

```

Notes: