Problem statement

The second problem (Exercise 2) asks us to design a Fraction class. Essentially, the characteristics of a fraction should be completely defined in order to have a proper Fraction object. Characteristics of a fraction is mentioned to be its ability to act as other numbers do.

It is also mentioned that the fractions are defined by the user and therefore, we must get an input from the user and consequently, design our program in a way that would be able to create a fraction object from the given input. Moreover, conversion from decimal to fraction vice versa as well as normalization is also required by the task.

Analysis

The input for this program would be an integer called choice that allows the user to travel around the user interface, integer numerators and denominators for the two input fractions and also a double integer for the input double to be converted to a fraction.

The output of this program can either be a fraction object, a string showing the result of a function or a double showing the result of fraction to double conversion and that is depending on the options the user decides to use.

There are several variables that need to be implemented in this program to carry out tasks such as storing the fractions' numerators and denominators, greatest common divisor, integer regarding users' program of choice (int choice) and a line, string stream and integer for the decimal input that are implemented the same way as exercise 1.

Design

As the first step of the algorithm, a menu (user interface) was designed to simplify users' control over the flow of the program. The menu consists of 7 different options that the users can choose from depending on their demands.

Note: There are three functions that are used and are essential in the following sections and therefore, are explained respectively below:

1) **gcdfinder Function ==>** here we define an integer i and then increase it accordingly until we find the greatest common divisor between the fraction's numerator and denominator.

2) **showFraction Function ==>** here, a fraction's top and bottom (numerator and denominator) are separated and printed as a string in the form top / bottom.

3) **normalise Function ==>** As tasked in the coursework task sheet, this function was designed to only allow the numerator to be negative and prevent the denominator from ever being negative. Moreover, we use gcdfinder here to make sure that the input fraction is in its least common denominator form.

For the first four parts, there are two functions designed to receive the first and second fraction from the user respectively. In these functions, it is mentioned that the user cannot input 0 as a denominator and if so, the denominator would be automatically set to 1 (For the fourth case, the same thing also happens to the second fraction's numerator as that cannot be 0 as well). Then the received fractions are sent to each of the following four sections (sections 1-4).

Below are provided the algorithm for the functions of the user interface:

1. **Find the sum**

   The input fractions are received. Then they are sent to the function called plus. Here the mathematical rules of sum of fractions are employed in order to find the result's numerator and denominator, meaning that the denominator was resulted from the multiplication of the two denominators while the numerator was made by multiplying the first fraction's numerator with the second fraction's denominator and adding that to the multiplication of the first fraction's denominator and the second fraction's numerator. Then using the two, a result fraction object is created and returned to the call. Consequently, the result is normalized using the normalise function and displayed by showFraction function. As the final part, the fraction is normalized and returned.

2. **Find the difference**

   The process is the same as 1. Except the fractions are sent to the function called minus rather than plus and there the multiplications regarding the calculation of the numerator are subtracted from each other rather than added together. As the final part, the fraction is normalized and returned.

3. **Find the multiplication**

   The fractions are sent to the function called times rather than plus or minus. In the times function, the numerators are multiplied with each other to form the result's numerator and the denominators are multiplied with each other to form the result's denominator. As the final part, the fraction is normalized and returned.

4. **Find the division**

   The process is the same as before. Except the first fraction's numerator is multiplied with the second fraction's denominator to form the result's numerator and the first fraction's denominator is multiplied by the second fraction's numerator to form the result's denominator. As the final part, the fraction is normalized and returned.

5. **Convert Fraction to Decimal**

   For comparing a fraction to decimal, we first send the fraction to the convertToDecimal function. There we divide the fractions numerator by its denominator and cast it to be a double value. This double value would be the decimal form of the given fraction.

6. **Convert Decimal to Fraction**

   At first, we receive a decimal as a string variable. Then the string is sent to the function called convertToFraction. First, the string is split into two parts: before the dot and after the dot. The number of numbers after the dot is saved to indicate the power of ten in the result's denominator. The result's numerator is constructed by adding the two string parts together and then converting them into an integer using a string stream. As the final part, the fraction is normalized and returned.

7. **Compare two fractions**

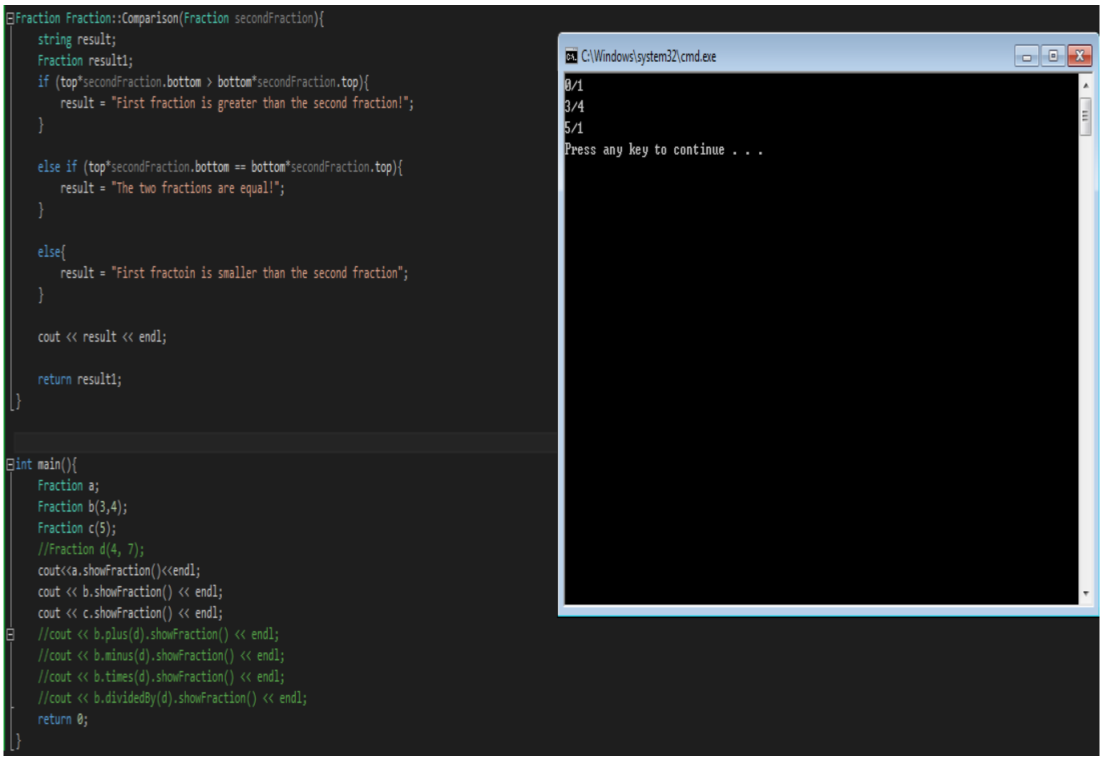
   We send to fractions to the comparedTo function. There we compare the values of a fraction’s numerator multiplied by the other fraction’s denominator. The fraction with the higher value is considered to be greater. If the values are equal, then the fractions are equal. A message is designed to be prompted to show the relation between the values (greater- equal- smaller).

Implementation

Name of the file regarding this exercise ==> Exercise2

Testing

For the first step of the test, three fractions are initialized as shown in the coursework task sheet to test the essential part of creating a fraction object (Figure 1).



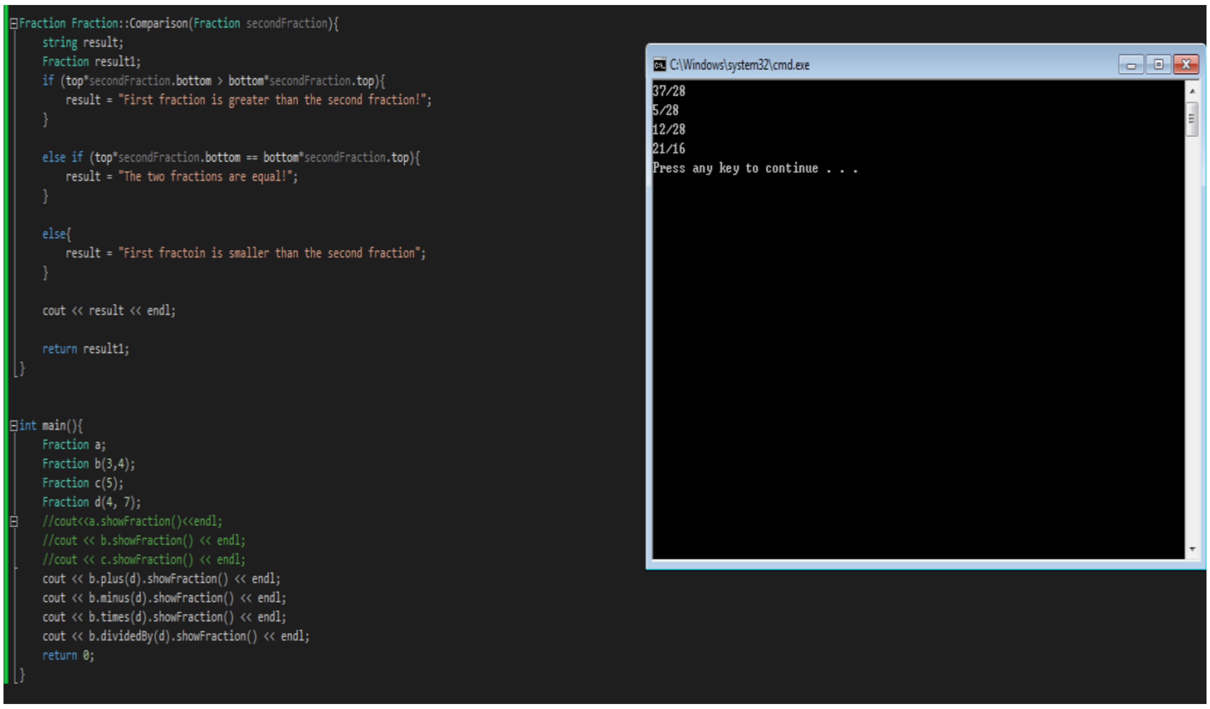
```
Fraction Fraction::Comparison(Fraction secondFraction){
    string result;
    Fraction result1;
    if (top*secondFraction.bottom > bottom*secondFraction.top){
        result = "First fraction is greater than the second fraction!";
    }

    else if (top*secondFraction.bottom == bottom*secondFraction.top){
        result = "The two fractions are equal!";
    }

    else{
        result = "First fractoin is smaller than the second fraction";
    }

    cout << result << endl;

    return result1;
}


int main(){
    Fraction a;
    Fraction b(3,4);
    Fraction c(5);
    //Fraction d(4, 7);
    cout<<a.showFraction()<<endl;
    cout << b.showFraction() << endl;
    cout << c.showFraction() << endl;
    //cout << b.plus(d).showFraction() << endl;
    //cout << b.minus(d).showFraction() << endl;
    //cout << b.times(d).showFraction() << endl;
    //cout << b.dividedBy(d).showFraction() << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
0/1
3/4
5/1
Press any key to continue . . .
```

Figure 1: the initial test (Fraction Object)

For the second step of testing, the declared functions were used in order to test whether the four functions plus, minus, times and divided by functioned properly or not (Figure 2).



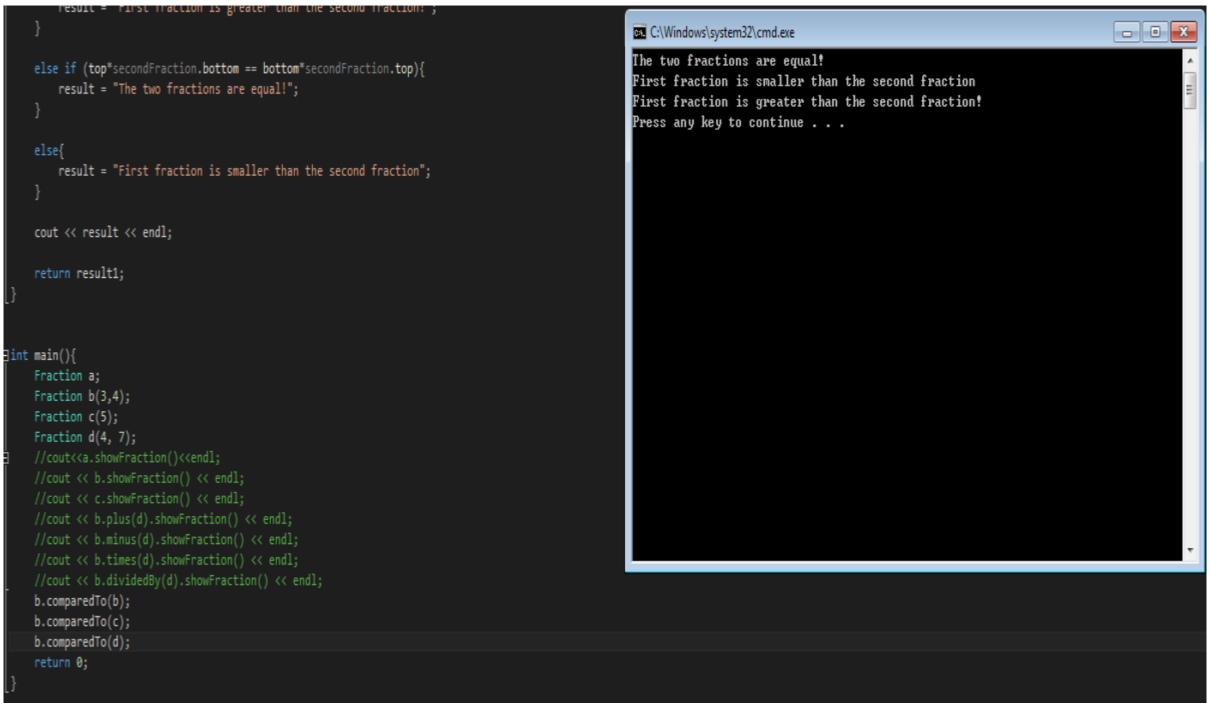
```cpp
Fraction Fraction::Comparison(Fraction secondFraction){
    string result;
    Fraction result1;
    if (top*secondFraction.bottom > bottom*secondFraction.top){
        result = "First fraction is greater than the second fraction!";
    }

    else if (top*secondFraction.bottom == bottom*secondFraction.top){
        result = "The two fractions are equal!";
    }

    else{
        result = "First fractoin is smaller than the second fraction";
    }

    cout << result << endl;

    return result1;
}


int main(){
    Fraction a;
    Fraction b(3,4);
    Fraction c(5);
    Fraction d(4, 7);
    //cout<<a.showFraction()<<endl;
    //cout << b.showFraction() << endl;
    //cout << c.showFraction() << endl;
    cout << b.plus(d).showFraction() << endl;
    cout << b.minus(d).showFraction() << endl;
    cout << b.times(d).showFraction() << endl;
    cout << b.dividedBy(d).showFraction() << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
37/28
5/28
12/28
21/16
Press any key to continue . . .
```

Figure 2: The second test (add, subtract, multiplication and division test)

Then the three states of comparison were tested in order to assure that the comparedTo function is properly working (Figure 3).



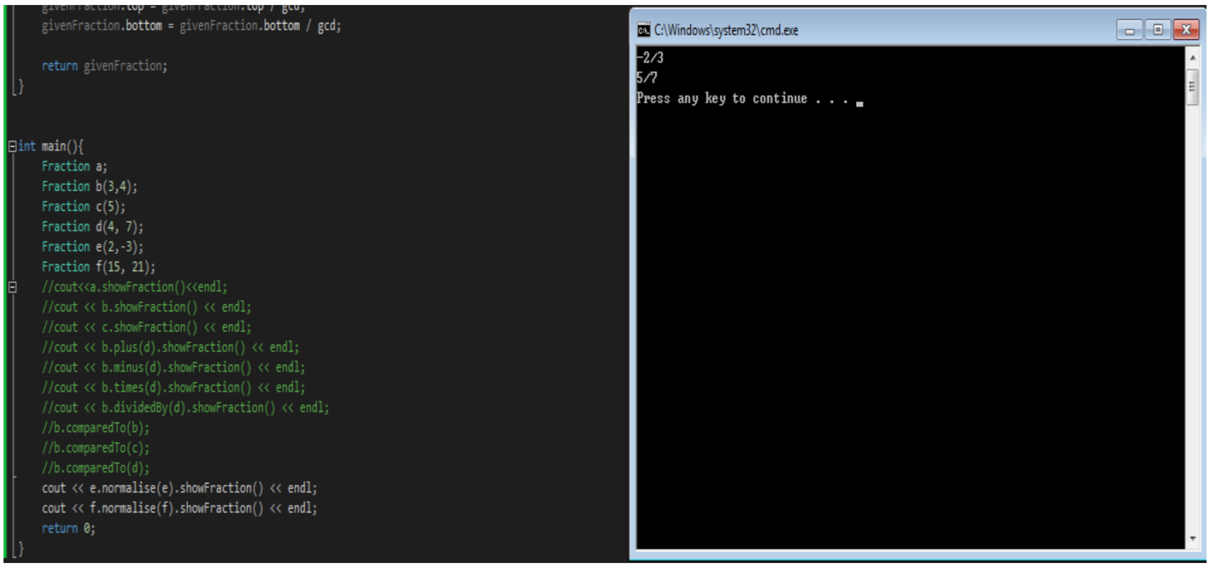
```cpp
        result = "First fraction is greater than the second fraction!";
    }

    else if (top*secondFraction.bottom == bottom*secondFraction.top){
        result = "The two fractions are equal!";
    }

    else{
        result = "First fraction is smaller than the second fraction";
    }

    cout << result << endl;

    return result1;
}


int main(){
    Fraction a;
    Fraction b(3,4);
    Fraction c(5);
    Fraction d(4, 7);
    //cout<<a.showFraction()<<endl;
    //cout << b.showFraction() << endl;
    //cout << c.showFraction() << endl;
    //cout << b.plus(d).showFraction() << endl;
    //cout << b.minus(d).showFraction() << endl;
    //cout << b.times(d).showFraction() << endl;
    //cout << b.dividedBy(d).showFraction() << endl;
    b.comparedTo(b);
    b.comparedTo(c);
    b.comparedTo(d);
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
The two fractions are equal!
First fraction is smaller than the second fraction
First fraction is greater than the second fraction!
Press any key to continue . . .
```

Figure 3: The comparison test

As the next step of the test, fractions with negative denominators were initialized in order to test normalise function’s functionality (Figure 4).



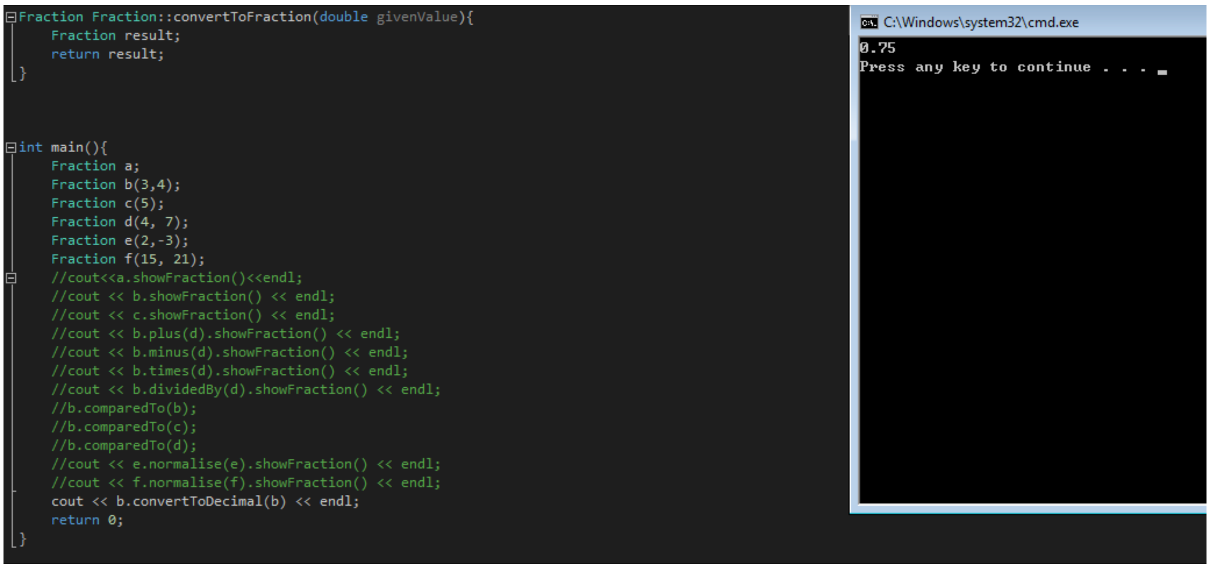
```
    givenFraction.top = givenFraction.top / gcd;
    givenFraction.bottom = givenFraction.bottom / gcd;

    return givenFraction;
}


int main(){
    Fraction a;
    Fraction b(3,4);
    Fraction c(5);
    Fraction d(4, 7);
    Fraction e(2,-3);
    Fraction f(15, 21);
    //cout<<a.showFraction()<<endl;
    //cout << b.showFraction() << endl;
    //cout << c.showFraction() << endl;
    //cout << b.plus(d).showFraction() << endl;
    //cout << b.minus(d).showFraction() << endl;
    //cout << b.times(d).showFraction() << endl;
    //cout << b.dividedBy(d).showFraction() << endl;
    //b.comparedTo(b);
    //b.comparedTo(c);
    //b.comparedTo(d);
    cout << e.normalise(e).showFraction() << endl;
    cout << f.normalise(f).showFraction() << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
-2/3
5/7
Press any key to continue . . .
```

Figure 4: The normalization test

Consequently, the following two tests were conducted to ensure the functionality of convertToDecimal and convertToFraction functions (Figures 5&6 respectively).

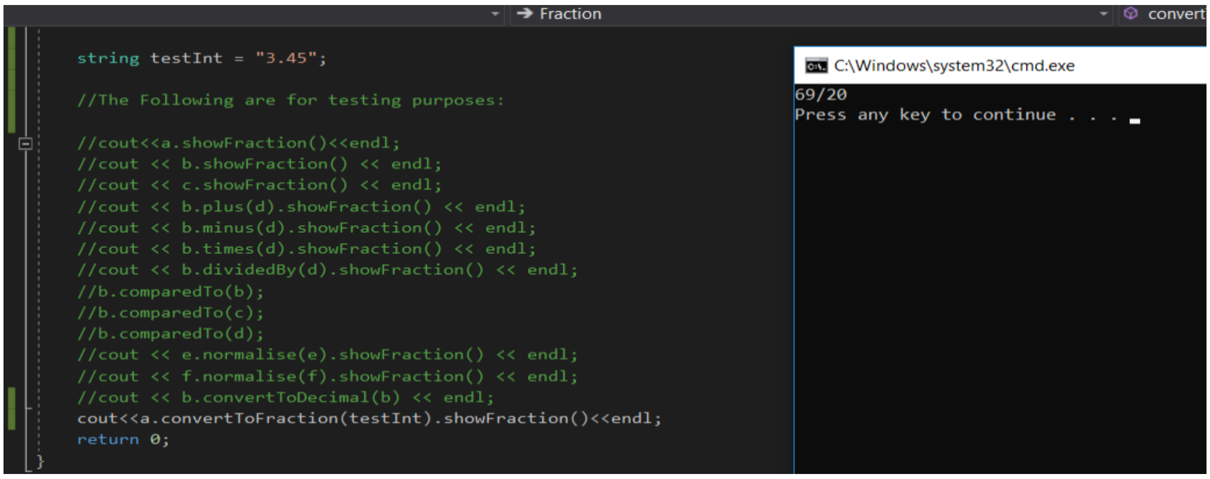


```
Fraction Fraction::convertToFraction(double givenValue){
    Fraction result;
    return result;
}


int main(){
    Fraction a;
    Fraction b(3,4);
    Fraction c(5);
    Fraction d(4, 7);
    Fraction e(2,-3);
    Fraction f(15, 21);
    //cout<<a.showFraction()<<endl;
    //cout << b.showFraction() << endl;
    //cout << c.showFraction() << endl;
    //cout << b.plus(d).showFraction() << endl;
    //cout << b.minus(d).showFraction() << endl;
    //cout << b.times(d).showFraction() << endl;
    //cout << b.dividedBy(d).showFraction() << endl;
    //b.comparedTo(b);
    //b.comparedTo(c);
    //b.comparedTo(d);
    //cout << e.normalise(e).showFraction() << endl;
    //cout << f.normalise(f).showFraction() << endl;
    cout << b.convertToDecimal(b) << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
0.75
Press any key to continue . . .
```

Figure 5: The convertToDecimal test

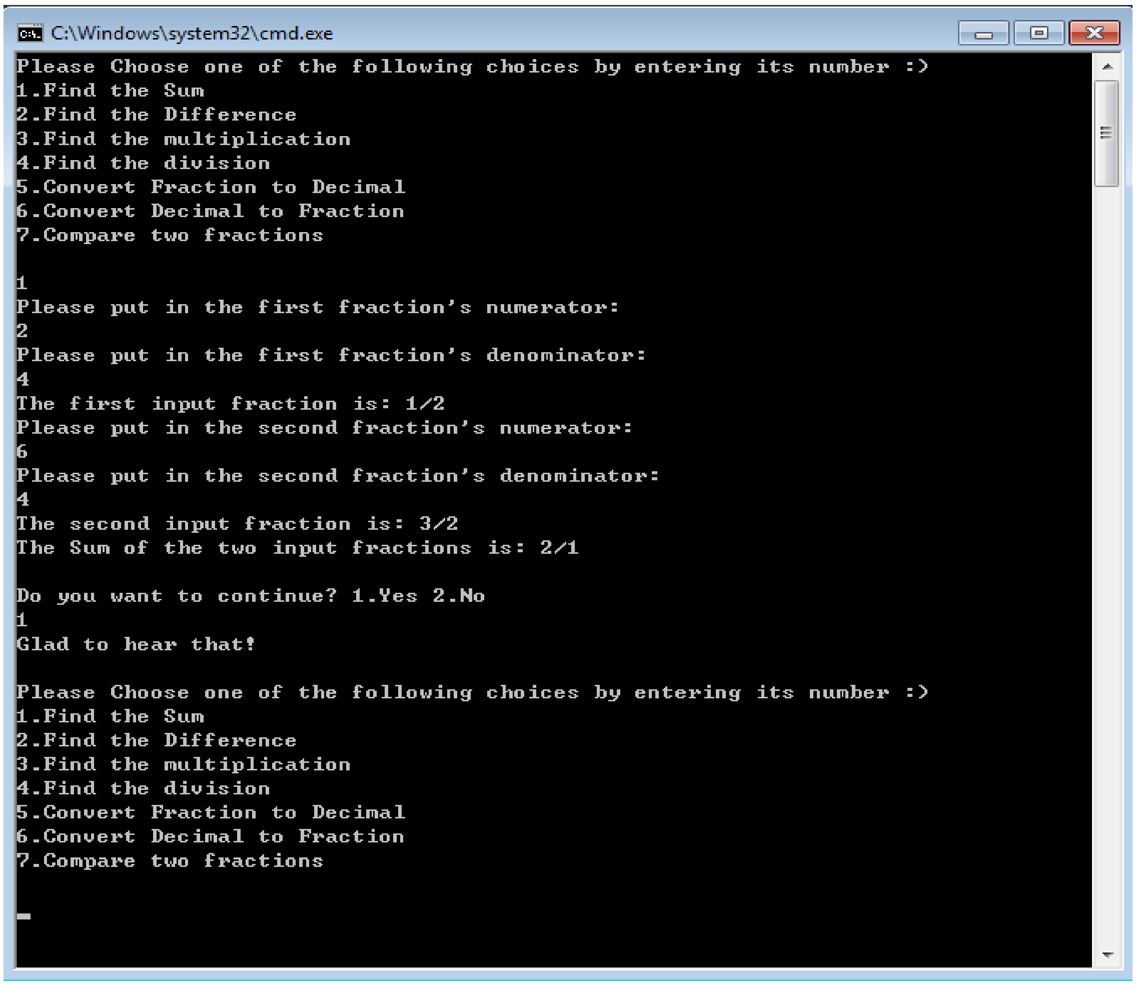


```
    string testInt = "3.45";

    //The Following are for testing purposes:

    //cout<<a.showFraction()<<endl;
    //cout << b.showFraction() << endl;
    //cout << c.showFraction() << endl;
    //cout << b.plus(d).showFraction() << endl;
    //cout << b.minus(d).showFraction() << endl;
    //cout << b.times(d).showFraction() << endl;
    //cout << b.dividedBy(d).showFraction() << endl;
    //b.comparedTo(b);
    //b.comparedTo(c);
    //b.comparedTo(d);
    //cout << e.normalise(e).showFraction() << endl;
    //cout << f.normalise(f).showFraction() << endl;
    //cout << b.convertToDecimal(b) << endl;
    cout<<a.convertToFraction(testInt).showFraction()<<endl;
    return 0;
}
```

```
69/20
Press any key to continue . . .
```

Figure 6: The convertToFraction test

And the last part of the test was dedicated to testing the user interface and using its features. In this part every option available in the provided menu was tested multiple times with different values and input to ensure complete functionality.



```
Please Choose one of the following choices by entering its number :)
1.Find the Sum
2.Find the Difference
3.Find the multiplication
4.Find the division
5.Convert Fraction to Decimal
6.Convert Decimal to Fraction
7.Compare two fractions

1
Please put in the first fraction's numerator:
2
Please put in the first fraction's denominator:
4
The first input fraction is: 1/2
Please put in the second fraction's numerator:
6
Please put in the second fraction's denominator:
4
The second input fraction is: 3/2
The Sum of the two input fractions is: 2/1

Do you want to continue? 1.Yes 2.No
1
Glad to hear that!

Please Choose one of the following choices by entering its number :)
1.Find the Sum
2.Find the Difference
3.Find the multiplication
4.Find the division
5.Convert Fraction to Decimal
6.Convert Decimal to Fraction
7.Compare two fractions
```

Figure 7: The user interface test

Note: It should be noted that due to prevention of redundancy, many of the test screenshots were not included inside this report as it is believed they all show the same outcome. There were numerous possibilities and combinations tested for the algorithm and user interface to ensure and they both work properly and swiftly.