Digital Assignment – 2

Object Oriented Programming

Name: Hari Krishna Shah

VIT ID: 21BCS0167

Ques 1. 1. Construct a class Fraction to represent a fraction of the type 3/8.

Overload = = to compare two fraction objects.

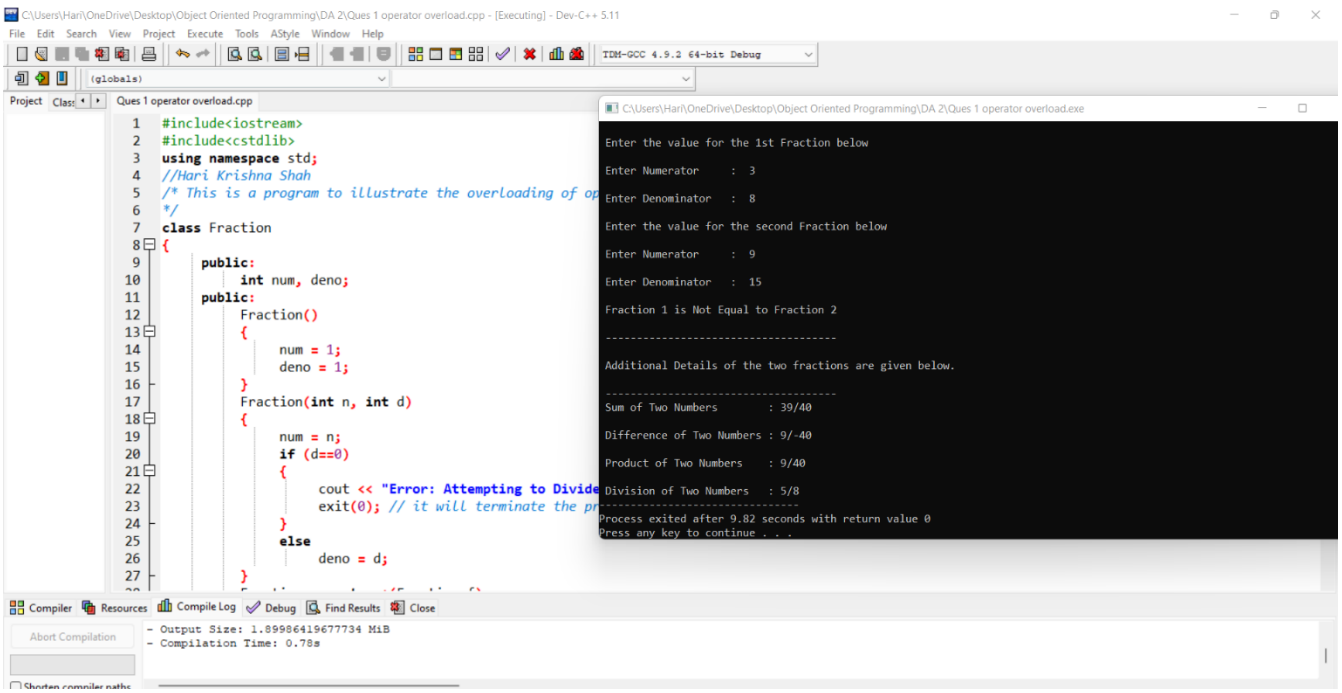
Answer:

```
#include<iostream>
#include<cstdlib>
using namespace std;
//Hari Krishna Shah
/* This is a program to illustrate the overloading of
operators.
*/
class Fraction
{
	public:
		int num, deno;
	public:
		Fraction()
		{
			num = 1;
			deno = 1;
		}
		Fraction(int n, int d)
		{
			num = n;
			if (d==0)
			{
				cout << "Error: Attempting to Divide by
```

```
Zero" << endl;
                    exit(0); // it will terminate the program
if division by 0 is attempted
               }
               else
                    deno = d;
          }
          Fraction operator +(Fraction f)
          {
               int n = num*f.deno+f.num*deno;
               int d = deno*f.deno;
               return Fraction(n/gcd(n,d),d/gcd(n,d));
          }
          Fraction operator -(Fraction f)
          {
               int n = num*f.deno-f.num*deno;
               int d = deno*f.deno;
               return Fraction(n/gcd(n,d),d/gcd(n,d));
          }
          Fraction operator *(Fraction f)
          {
               int n = num*f.num;
               int d = deno*f.deno;
               return Fraction(n/gcd(n,d),d/gcd(n,d));
          }
          Fraction operator /(Fraction f)
          {
               int n = num*f.deno;
               int d = deno*f.num;
               return Fraction(n/gcd(n,d),d/gcd(n,d));
          }
          bool operator == (Fraction &f)
          {
               return (num==f.num) && (deno==f.deno);
          }
          int gcd(int n, int d)
          {
               int rem;
               while (d != 0)
               {
                    rem = n % d;
                    n = d;
                    d = rem;
               }
               return n;
```

```
        }
        void accept()
        {
            cout<<"\n Enter Numerator     :  ";
            cin>>num;
            cout<<"\n Enter Denominator   :  ";
            cin>>deno;
        }
};
int main()
{
    cout<<" Welcome to the Digital Assignment - 2 of Hari
Krishna Shah."<<endl;
     class Fraction f[3];

    cout<<"\n Enter the value for the 1st Fraction below\n";

    f[1].accept();

    cout<<"\n Enter the value for the second Fraction
below\n";
    f[2].accept();

    if(f[1] == f[2])
        cout<<"\n Fraction 1 is Equal to Fraction 2"<<endl;
    else
        cout<<"\n Fraction 1 is Not Equal to Fraction
2"<<endl;


    //Overloading other arthimetic Operators

    cout<<"\n ------------------------------------"<<endl;
    cout<<"\n Additional Details of the two fractions are
given below."<<endl;

    f[3]=f[1]+f[2];
    cout<<"\n ------------------------------------";
    cout<<"\n Sum of Two Numbers        :
"<<f[3].num<<"/"<<f[3].deno<<endl;

    f[3]=f[1]-f[2];
    cout<<"\n Difference of Two Numbers :
"<<f[3].num<<"/"<<f[3].deno<<endl;
```

```
	f[3]=f[1]*f[2];
	cout<<"\n Product of Two Numbers    :
"<<f[3].num<<"/"<<f[3].deno<<endl;

	f[3]=f[1]/f[2];
	cout<<"\n Division of Two Numbers   :
"<<f[3].num<<"/"<<f[3].deno;


	return 0;
}
```



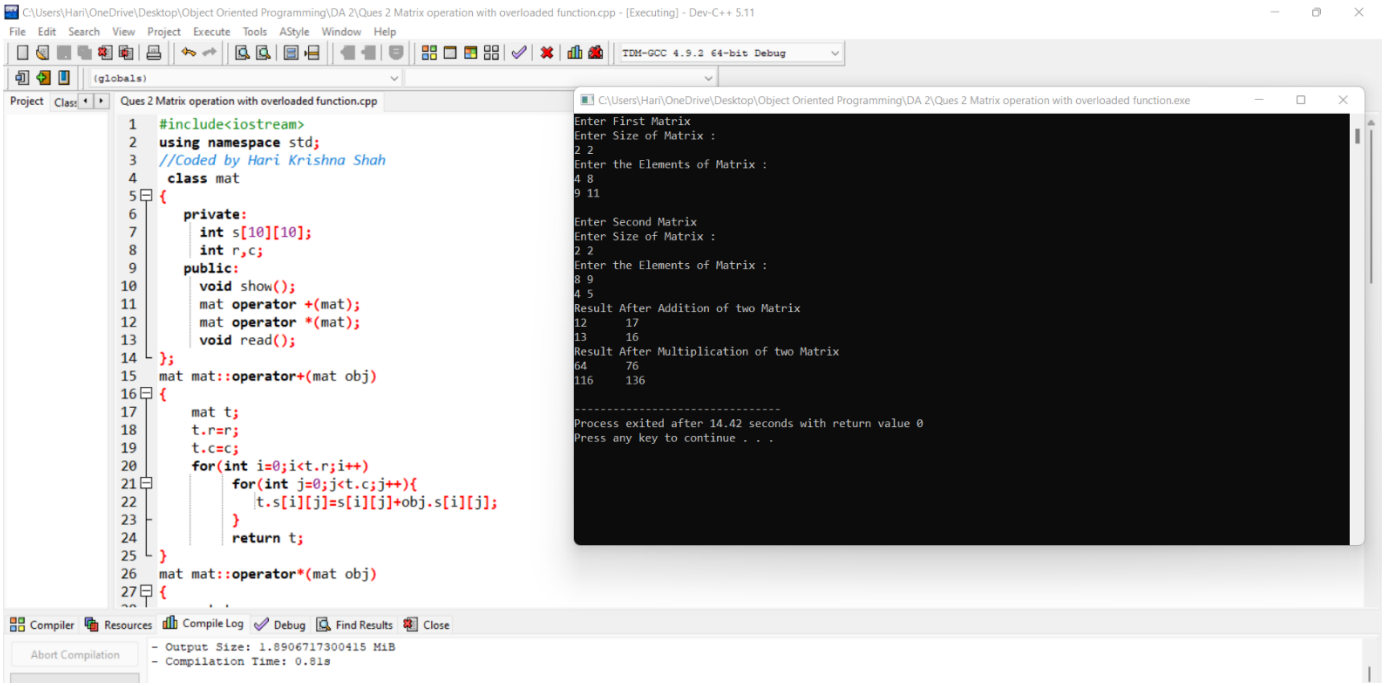
Ques 2. Develop an OOP to perform the Matrix addition and multiplication by overloading addition operator + and multiplication operator * .

Answer:

```
#include<iostream>
using namespace std;
//Coded by Hari Krishna Shah
 class mat
{
```

```
   private:
     int s[10][10];
     int r,c;
   public:
     void show();
     mat operator +(mat);
     mat operator *(mat);
     void read();
};
mat mat::operator+(mat obj)
{
    mat t;
    t.r=r;
    t.c=c;
    for(int i=0;i<t.r;i++)
         for(int j=0;j<t.c;j++){
          t.s[i][j]=s[i][j]+obj.s[i][j];
           }
           return t;
}
mat mat::operator*(mat obj)
{
    mat t;
    t.r=r;
    t.c=obj.c;
    for(int i=0;i<t.r;i++){
     for(int j=0;j<t.c;j++)
        {
             t.s[i][j]=0;
             for(int k=0;k<c;k++){
               t.s[i][j]+=s[i][k] * obj.s[k][j];
               }

        }
     }

    return t;
}
void mat::read()
{
   cout<<"Enter Size of Matrix : \n";
   cin>>r>>c;
   cout<<"Enter the Elements of Matrix :\n";
   for(int i=0;i<r;i++){
          for(int j=0;j<c;j++){
```

```
            cin>>s[i][j];
            }

    }

}
void mat::show()
{
  for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
            cout<<s[i][j]<<"\t";
           }
        cout<<"\n";
     }
}
int main()
{
      mat obj1 ,obj2,obj3;
      cout<<"Enter First Matrix\n";
      obj1.read();
       cout<<endl;
      cout<<"Enter Second Matrix\n";
      obj2.read();
      obj3=obj1 + obj2;
      cout<<"Result After Addition of two Matrix\n";
      obj3.show();
      obj3=obj1 * obj2;
      cout<<"Result After Multiplication of two Matrix\n";
      obj3.show();
}
```



Ques 3. Operator function can be defined using member function and friend function. Compare it. Why some of the operators cannot be overloaded using friend function but possible with member function? Reason out.

Answer:

The difference between operator function using member function and friend function is given below.

The main difference is that friend function requires arguments to explicitly passed them to the function and process explicitly whereas, member function considers the first argument implicitly.

Operator overloading of member function

Member functions are operators and functions declared as members of a certain class. They don't include operators and functions declared with the friend keyword.

If you write an operator function as a member function, it gains access to all of the member variables and functions of that class.

When overloading an operator using a member function:

- The overloaded operator must be added as a member function of the left operand.
- The left operand becomes the implicit object
- All other operands become function parameters.

Example:

```
#include <iostream>

class Coins
{
private:
   int a_coins;

public:
   Coins(int coins) { a_coins = coins; }

   // Overload Coins + int
   Coins operator+(int value);

   int getCoins() const { return a_coins; }
};
```

Output:
We have 9 coins.

Operator overloading using friend function

A non-member function does not have access to the private data of that class.

This means that an operator overloading function must be made a friend function if it requires access to the private members of the class.

Example:

```
#include <iostream>

class Coins
{
private:
	int a_coins;

public:
	Coins(int coins) { a_coins = coins; }

	// add Coins + Coins using a friend function
	friend Coins operator+(const Coins &c1, const Coins &c2);

	int getCoins() const { return a_coins; }
};

// note: this function is not a member function!
Coins operator+(const Coins &c1, const Coins &c2)
{
	// use the Coins constructor and operator+(int, int)
```

```
    // we can access a_coins directly because this is a friend function
    return Coins(c1.a_coins + c2.a_coins);
}

int main()
{
    Coins coins1{ 5 };
    Coins coins2{ 4 };
    Coins coinsSum{ coins1 + coins2 };
    std::cout << "We have " << coinsSum.getCoins() << " coins.\n";

    return 0;
}
```

Output:
We have 9 coins.

The classes istream and ostream doesn't defined in the header file iostream.h doesn't allow to overload the following operator using friend function but they can be overloaded using member function.

If the following operators are overloaded using friend function, then program will result with compilation error. Friend function is a function that can access the data from private, protected and public class.

- function call operator ()
- assignment operator =
- class member access operator ->
- subscripting operator [ ]