Project 3

CSCI 140

Jin Y Choi
Email - jchoi101@student.mtsac.edu

November 5th, 2018

Development Environment
System - PC Ubuntu 16.04
Compiler: CLion

Table of Contents

Program Notes:
The status of the program is 100% complete, all required test cases tested and working and all functions implemented. Input validation is not perfect for the program, if there are extra spaces or unexpected characters in places where the program does not expect it, the program may crash (ie. if the input expression is 1/2- + 1, it would incorrectly read in the second operand because of the '-' char). However if the input is in the form of w, w n/d or n/d the input works correctly, and as a result the operations run correctly as well.

Besides very specific input validation, the program works fully as intended, however the overall organization of the code is sloppy, as more and more exception handling had to be done that deviated from the initially outlined pseudocode.

Extra Credit 2 was implemented. When the program calls the sentinel expression (0/1 % 0/1) it outputs the answers that were answered incorrectly followed by operator that had the most incorrect answers (shown in test case).

I have implemented extra credit 2 in such a way that if there is a tie between the count for the operations that resulted in the most error, it will output all the operators (shown in test case, there is 1 problem wrong from addition, subtraction and multiplication and it will output all three operators.)

In addition to required features for the project, I have also implemented the istream friend operator>> overload function to actual be able to take in whole number inputs (which will then assume that the parameters not inputted default to the 0 and 1 values for the numerator and denominator respectively). This is shown in the second test case.

Source Code:

```
/*  Program: MFraction.h
   Author: Jin Choi
   Class: CSCI 140
   Date: 11/5/2018
   Description: The header file for the MFraction class.

   I certify that the code below is my own work.

  Exception(s): N/A

*/

#ifndef PROJECT_3_MFRACTION_H
#define PROJECT_3_MFRACTION_H

#include <iostream>
#include <string>

using namespace std;

class MFraction {
private:
   int numerator;
   int denominator;
   int whole;
   MFraction simplify(int, int, int) const;
   MFraction simplifySigns(int, int, int) const;
   int GCD(int a, int b) const;
public:
   MFraction(int n = 0, int d = 1,int w = 0);
   int getDenominator() const;
   int getNumerator() const;
   int getWhole() const;
   MFraction& setNumerator(int n);
   MFraction& setDenominator(int d);
   MFraction& setWhole(int w);
   MFraction operator+ (const MFraction &s) const;
   MFraction operator- (const MFraction &s) const;
   MFraction operator* (const MFraction &s) const;
   MFraction operator/ (const MFraction &s) const;
   MFraction operator- () const;
   void printFloat() const;
   bool operator<(const MFraction &s) const;
   bool operator<= (const MFraction &s) const;
   bool operator> (const MFraction &s) const;
   bool operator>= (const MFraction &s) const;
   bool operator== (const MFraction &s) const;
   bool operator!= (const MFraction &s) const;
   friend istream& operator>> (istream &in, MFraction &f);
```

```
    friend ostream& operator<< (ostream &out, const MFraction &f);
};


#endif //PROJECT_3_MFRACTION_H

/*  Program: main.cpp
    Author: Jin Choi
    Class: CSCI 140
    Date: 11/5/2018
    Description: The driver file for the MFraction class that executes the program.

    I certify that the code below is my own work.

   Exception(s): N/A

*/

#include "MFraction.h"
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>

struct Pair{
    char oper;
    int count;
};



int main() {
    cout << "Jin Choi's Mixed Fractions Tutorial Program" << endl << endl;
    cout << "Please follow instructions carefully.\nEnter your operation like 1 1/2 + 1/4.\nEnter 0/1 % 0/1 to stop
the program." << endl << endl;
    int correct = 0, total = 0;
    struct Pair signs[4] = {
            {'+',0},
            {'-',0},
            {'*',0},
            {'/',0}
    };
    vector<MFraction> oper1;
    vector<char> operators;
    vector<MFraction> oper2;
    MFraction frac1, frac2, frac3;
    MFraction response;
    char oper;
    while(true){
        cout << setw(28) << "Please enter your operation " << "--> ";
```

```
        cin >> frac1;
        cin >> oper;
        cin >> frac2;
        bool valid = true;
        if (frac1.getDenominator() == 0){
            cout << frac1 << " is an invalid operand" << endl << endl;
            valid = false;
        }
        if (frac2.getDenominator() == 0){
            cout << frac2 << " is an invalid operand" << endl << endl;
            valid = false;
        }
        if (oper == '/' && frac2.getNumerator() == 0){
            cout << "Invalid expression, division by 0." << endl << endl;
            valid = false;
        }
        if (frac1.getWhole() == 0 && frac1.getNumerator()==0 && frac1.getDenominator()==1 && oper == '%' &&
frac2.getWhole()==0 && frac2.getNumerator()==0 && frac2.getDenominator()==1) {
            cout << "You answer " << correct << " out of " << total << " questions correctly.\n\n";
            cout << "The problems with incorrect answers:\n";
            int n = oper1.size();
            for (int i = 0 ; i < n;i++){
                cout << oper1.back() << " " << operators.back() << " " << oper2.back();
                oper1.pop_back();
                operators.pop_back();
                oper2.pop_back();
                cout << endl;
            }
            int max = 0;
            for (int i = 0; i < 3; i++){
                max = signs[max].count < signs[i].count ? i : max;
            }
            cout << endl << "It seems like you have the most problem with " << signs[max].oper << "operations.";
            break;
        }if (valid){
            switch (oper){
                case '+': frac3 = frac1 + frac2; break;
                case '-': frac3 = frac1 - frac2; break;
                case '*': frac3 = frac1 * frac2; break;
                case '/': frac3 = frac1 / frac2; break;
                default: cout << oper << " is an invalid operator" << endl << endl; valid = false; break;
            }
        }
        if (valid){
            cout << setw(28) << left << "Please enter your result" << "--> ";
            cin >> response;
            if (frac3 == response){
                cout << "Congratulations! It is correct." << endl;
                correct++;
            }
```

```
            else{
                oper1.push_back(frac1);
                operators.push_back(oper);
                oper2.push_back(frac2);
                switch (oper){
                    case '+': signs[0].count++; break;
                    case '-': signs[1].count++; break;
                    case '*': signs[2].count++; break;
                    case '/': signs[3].count++; break;
                    default: break;
                }
                cout << "It is incorrect. The correct answer is" << endl;
                cout << frac1 << " " << oper << " " << frac2 << " = " << frac3 << endl;
            }
            frac3.printFloat();
            total++;
        }
        cin.clear();
        cin.ignore(INTMAX_MAX, '\n');
    }
    return 0;
}

/*  Program: MFraction.cpp
    Author: Jin Choi
    Class: CSCI 140
    Date: 11/5/2018
    Description: The implementation file for the MFraction class.

    I certify that the code below is my own work.

   Exception(s): N/A

*/

#include "MFraction.h"
#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;

MFraction::MFraction(int n, int d, int w){
    whole = w;
    numerator = n;
    denominator = d;
}

int MFraction::getDenominator() const { return denominator;}
int MFraction::getNumerator() const { return numerator; }
```

```
int MFraction::getWhole() const { return whole; }

MFraction &MFraction::setNumerator(int n){
  numerator = n;
  return *this;
}
MFraction &MFraction::setDenominator(int d){
  denominator = d;
  return *this;
}
MFraction &MFraction::setWhole(int w){
  whole = w;
  return *this;
}
MFraction MFraction::operator+ (const MFraction &s) const{
  int common = denominator*s.denominator;
  int num = 0;
  if (whole < 0){
     num += (whole*denominator-numerator)*s.denominator;
  }
  else{
     num += (whole*denominator+numerator)*s.denominator;
  }
  if (s.whole < 0){
     num += (s.whole*s.denominator-s.numerator)*denominator;
  }
  else{
     num += (s.whole*s.denominator+s.numerator)*denominator;
  }
  return simplify(num,common, 0);
}

MFraction MFraction::operator- (const MFraction &s) const{
  int common = denominator*s.denominator;
  int num = 0;
  if (whole < 0){
     num += (whole*denominator-numerator)*s.denominator;
  }
  else{
     num += (whole*denominator+numerator)*s.denominator;
  }
  if (s.whole < 0){
     num -= (s.whole*s.denominator-s.numerator)*denominator;
  }
  else{
     num -= (s.whole*s.denominator+s.numerator)*denominator;
  }
  return simplify(num, common, 0);
}
MFraction MFraction::operator* (const MFraction &s) const{
```

```
  int num = (whole * denominator + numerator) * (s.whole * s.denominator + s.numerator);
  int denom = denominator * s.denominator;
  return simplify(num,denom,0);
}
MFraction MFraction::operator/ (const MFraction &s) const{
  int num = (whole * denominator + numerator) * (s.denominator);
  int denom = (s.whole * s.denominator + s.numerator) * denominator;
  return simplify(num,denom,0);
}
MFraction MFraction::operator- () const{ // Negation
  MFraction result;
  if (whole == 0){
     result = simplifySigns(-numerator,denominator, whole);
  }else{
     result = simplifySigns(numerator,denominator,-whole);
  }
  return result;
}
void MFraction::printFloat() const{
  int num = numerator + whole*denominator;
  cout << "Its floating-point value is " << static_cast<double>(num)/denominator << endl << endl;
}
bool MFraction::operator< (const MFraction &s) const{
  if (whole < s.whole){
     return true;
  }
  else if (whole > s.whole){
     return false;
  }
  else{
     int common = denominator * s.denominator;

     return (numerator * common) < (s.numerator * common);
  }
}
bool MFraction::operator<= (const MFraction &s) const{
  if (whole < s.whole){
     return true;
  }
  else if (whole > s.whole){
     return false;
  }
  else{
     int common = denominator * s.denominator;

     return (numerator * common) <= (s.numerator * common);
  }
}
bool MFraction::operator> (const MFraction &s) const{
  if (whole > s.whole){
```

```
        return true;
    }
    else if (whole < s.whole){
        return false;
    }
    else{
        int common = denominator * s.denominator;

        return (numerator * common) > (s.numerator * common);
    }
}
bool MFraction::operator>= (const MFraction &s) const{
    if (whole > s.whole){
        return true;
    }
    else if (whole < s.whole){
        return false;
    }
    else{
        int common = denominator * s.denominator;

        return (numerator * common) >= (s.numerator * common);
    }
}
bool MFraction::operator== (const MFraction &s) const{
    int common = denominator * s.denominator;

    return whole == s.whole && ((numerator*common) == (s.numerator*common));
}

bool MFraction::operator!= (const MFraction &s) const{
    int common = denominator * s.denominator;

    return whole != s.whole || ((numerator*common) != (s.numerator*common));
}

istream& operator>> (istream &in, MFraction &f){
    f.setWhole(0).setNumerator(0).setDenominator(1);
    bool negate = false;
    if (in.peek() == 45){
        in.get();
        if (in.peek() == 32){
            negate = true;
        }
        else{
            in.putback(45);
        }
    }
    in >> f.whole;
    if (in.peek() != 10){
```

```
    if (in.peek() == 46){
        in.ignore();
    }
    if (in.peek() == 47){
        f.numerator = f.whole;
        f.whole = 0;
        in.ignore();
        in >> f.denominator;
    }
    else{
        char temp = in.get();
        char temp2;
        if (in.peek() == 42 || in.peek() == 43 || in.peek() == 45 || in.peek() == 47){
            temp2 = in.get();
            if (in.peek() == 32){
                in.putback(temp2);
                f.numerator = 0;
                f.denominator = 1;
            }else{
                in.putback(temp2);
                in.putback(temp);
                in >> f.numerator;
                in.ignore();
                in >> f.denominator;
            }
        }
        else{
            in.putback(temp);
            in >> f.numerator;
            in.ignore();
            in >> f.denominator;
        }
    }
}
if (negate){
    f = -f;
}
if (f.whole < 0 || f.whole > 0){
    if (f.numerator < 0 && f.denominator < 0){
        f.numerator = -f.numerator;
        f.denominator = -f.denominator;
    }
    else if(f.numerator < 0 ){
        f.whole = -f.whole;
        f.numerator = -f.numerator;
    }
    else if(f.denominator < 0){
        f.whole = -f.whole;
        f.denominator = -f.denominator;
    }
```

```
  }else{
     if ((f.numerator < 0 && f.denominator < 0)||(f.numerator > 0 &&f.denominator < 0)){
        f.numerator = -f.numerator;
        f.denominator = -f.denominator;
     }

  }

  return in;
}

ostream& operator<< (ostream &out, const MFraction &f){
  int w,n,d;
  w = f.whole;
  n = f.numerator;
  d = f.denominator;
  if (n != 0 && d != 0){
     if (abs(n) >= abs(d)){
        if (w < 0){ w -= n/d; }
        else{ w += n/d; }
        n = abs(n%d);
        d = abs(d);
     }
     if (n != 0){
        int GCM = f.GCD(n,d);
        n /= GCM;
        d /= GCM;
     }
  }
  if (w != 0){
     if (n != 0 ) {
        if ((n < 0 && d > 0) || (n > 0 && d < 0)){
           out << -w << " " << -n << "/" << d;
        }else{
           out << w << " " << n << "/" << d;
        }
     }
     else{
        out << w;
     }
  }else{
     if ( n != 0 ){
        if ((n < 0 && d > 0 )||(n > 0 && d > 0)){
           out << n << "/" << d;
        }
        else{
           out << -n << "/" << -d;
        }
     }else{
        out << w;
```

```
        }
    }
    return out;
}

int MFraction::GCD(int a, int b) const{
    if (abs(b) > abs(a)){
        int temp = b;
         b = a;
         a = temp;
    }
    if (a%b==0){
        return b;
    }
    return GCD(b,a%b);
}


MFraction MFraction::simplify(int n, int d, int w) const{
    if (n >= d){
        w += n/d;
        n = n%d;
    }
    if (n != 0){
        int common = GCD(n,d);
        n /= common;
        d /= common;
        return MFraction(n,d,w);
    }
    else{
        return MFraction(0,d,w);
    }
}

MFraction MFraction::simplifySigns(int numerator, int denominator, int whole) const {
    if (whole < 0 || whole > 0){
        if (numerator < 0 && denominator < 0){
            numerator = -numerator;
            denominator = -denominator;
        }
        else if(numerator < 0 ){
            whole = -whole;
            numerator = -numerator;
        }
        else if(denominator < 0){
            whole = -whole;
            denominator = -denominator;
        }
    }else{
        if ((numerator < 0 && denominator < 0)||(numerator > 0 &&denominator < 0)){
```

```
        numerator = -numerator;
        denominator = -denominator;
      }

    }
    return MFraction(numerator,denominator,whole);
}
```

Input/Output:

```
Jin Choi's Mixed Fractions Tutorial Program

Please follow instructions carefully.
Enter your operation like 1 1/2 + 1/4.
Enter 0/1 % 0/1 to stop the program.

Please enter your operation --> 1/2 + 1/4
Please enter your result    --> 3/4
Congratulations! It is correct.
Its floating-point value is 0.75

Please enter your operation --> 1. 1/2 - 0 1/4
Please enter your result    --> 1. 3/4
It is incorrect. The correct answer is
1 1/2 - 1/4 = 1 1/4
Its floating-point value is 1.25

Please enter your operation --> 1/2 * 1/4
Please enter your result    --> 1. 1/8
It is incorrect. The correct answer is
1/2 * 1/4 = 1/8
Its floating-point value is 0.125

Please enter your operation --> 1. 1/8 / 0/1
Invalid expression, division by 0.

Please enter your operation --> 1/2 / 1/4
Please enter your result    --> 2. 0/1
Congratulations! It is correct.
Its floating-point value is 2

Please enter your operation --> -  1/4 + 10 1/20
Please enter your result    --> -  9 4/5
It is incorrect. The correct answer is
-1/4 + 10 1/20 = 9 4/5
Its floating-point value is 9.8

Please enter your operation --> -  15 1/2 * 0/1
Please enter your result    --> 0 0/1
Congratulations! It is correct.
Its floating-point value is 0

Please enter your operation --> 0/1 / -1 1/4
Please enter your result    --> 0/1
Congratulations! It is correct.
Its floating-point value is -0

Please enter your operation --> 15 0/1 - 5 0/1
Please enter your result    --> 10 0/1
Congratulations! It is correct.
Its floating-point value is 10

Please enter your operation --> 5 1/0 * -2 3/5
5 1/0 is an invalid operand

Please enter your operation --> 4 1/2 % 3 1/4
% is an invalid operator

Please enter your operation --> 0/1 % 0/1
You answer 5 out of 8 questions correctly.

The problems with incorrect answers:
-1/4 + 10 1/20
1/2 * 1/4
1 1/2 - 1/4

It seems like you have the most problem with +, -, *  operation(s).
Process finished with exit code 0
```

```
Jin Choi's Mixed Fractions Tutorial Program

Please follow instructions carefully.
Enter your operation like 1 1/2 + 1/4.
Enter 0/1 % 0/1 to stop the program.

Please enter your operation --> 3 * 6
Please enter your result    --> 15
It is incorrect. The correct answer is
3 * 6 = 18
Its floating-point value is 18

Please enter your operation --> 3 * 6
Please enter your result    --> 18
Congratulations! It is correct.
Its floating-point value is 18

Please enter your operation --> 3 / 6
Please enter your result    --> 1/2
Congratulations! It is correct.
Its floating-point value is 0.5

Please enter your operation --> 3 + 6
Please enter your result    --> 9
Congratulations! It is correct.
Its floating-point value is 9

Please enter your operation --> 3 + 6
Please enter your result    --> 15
It is incorrect. The correct answer is
3 + 6 = 9
Its floating-point value is 9

Please enter your operation --> 3 - 6
Please enter your result    --> 3
It is incorrect. The correct answer is
3 - 6 = -3
Its floating-point value is -3

Please enter your operation --> 0/1 % 0/1
You answer 3 out of 6 questions correctly.

The problems with incorrect answers:
3 - 6
3 + 6
3 * 6

It seems like you have the most problem with +, -, *  operation(s).
Process finished with exit code 0
```