EEE102 Assessment 1 Report

Yimian Liu 1717608

March 13, 2019

1. Introduction

After having a basic understanding concerning the most efficient programming language C in last semester, this year we are honored to learn C++, which is object oriented and tend to make development and cooperation easier and enjoyable. As the first assignment of EEE102, this exercise aims to let us have a basic scope about what is a object oriented programming language and beginning to get used to Class, Object, Container (e.g. vector) and Quote (e.g. int&) etc. under the instruction of software development process(SDP).

In the development process, we are required to designed a function which can compare if two int type vectors are same and also to design a Fraction Class which provides usual fraction operation in C++. This will be more detailly discussed in the following paragraph.

2. Problem Statement

2.1 Vectors Comparison

There are mainly four steps for this problem according to the assignment requirement.

2.1.1 Get User's Input

Before comparing two vectors and test our function, we are suggested to design an input process which can automatically get input which ended with a return key from user's input behavior and temporarily store them somewhere.

2.1.2 Allocate Input Content to Vector

This process aims to get all int type number without any other illegal value from the former user's input and allocated them into vector appropriately.

2.1.3 Compare Two Vector

In this section, we need to use two different methods to compare two vectors, packages this process into a function and return whether these two vectors are same using bool type. In this scope, the order and repeat of elements in vectors are ignored. For example, vector {1, 2, 3, 3, 4} and vector {4, 4, 3, 2, 2, 1} are regarded as the same because for every element in the first vector you can find a same one in the second.

2.2 Fraction Class

This question requests us to define a new class which can represent a normal fraction and can deal with basic fraction operation such as add, divide etc. Requirements can be divided in to three groups, the basic one, the advanced one and my additional requirement under input, output and operation section.

2.2.1 Input

Basic Requirement

- The Fraction can be declared with nothing and means 0. E.g. Fraction a;
- The Fraction can be declared with both a numerator and denominator. E.g. Fraction b(3,-4);
- The Fraction can be declared with simple numerator. E.g. Fraction c(5);

Advanced Requirement

- The Fraction can be declared with another Fraction. E.g. Fraction f = b;
- Input is normalized and simplified.
- Only the numerator can be negative.
- Decimals should be converted to Fraction form.

My Requirement

- The Fraction can be declared with a double type number. E.g. Fraction d(-3.14);
- The Fraction can be declared with double number directly. E.g. Fraction e = 1.3333;
- Recognize repeated decimals and transfer it to corresponding Fraction.
- Can be input use cin with decimals. E.g. cin >> f; (input: 3.14)
- Can be input use cin with fraction. E.g. cin >> f; (input: -3/4)

2.2.2 Output

Basic Requirement

- There has an output.

Advanced Requirement

- Can output decimals. E.g. f.val();
- Can output numerator and denominator. E.g. f.top(), f.bottom();

My Requirement

- Can directly use cout to output.

2.2.3 Operation

Basic Requirement

- Add, subtract, multiple and divide. E.g. (+, -, *, /)
- Compare base on values. E.g. (==, <, <=, >, >=, !=).

My Requirement

- Get opposite number and reciprocal. E.g. Fraction f(3,4); -f = -3/4; ~f = 4/3;
- Get remainder. E.g. f%2;
- Allow f++ and ++f, f--and --f.
- Allow f +=?, f -=?, f *=?, f /=?.
- Can directly operate with other numbers.

3. Analysis

3.1 Vectors Comparison

3.1.1 Input

The function is suggested to have two int type vectors as its parameters. While the value of these vectors should be input by users via cin stream. For one input which ended with a return key, the delimiter between numbers can be anything other than numbers. Which may mean that, we need to config where a number begin and where it ends. Inevitably, the input from users would be anything thus all possible input such as ',%<J' must be considered.

3.1.2 Output

The output of the function is a bool type variable. Besides, this comparison result should be output to the screen with cout stream.

3.1.3 Variables

Corresponding with both input and output process, at least four variables are required such as a string type variable for temporarily store user's input, two int type vectors, and also an int variable for extract int number from string.

As for the variables in the vec_same() function, for each algorithm, different local variables are utilized.

3.2 Fraction Class

3.2.1 Variables

Due to the problem statement in last section, we decide to use three variables in the class to store a fraction. A bool to store whether the fraction is negative. Two unsigned long int to store the numerator and denominator. All of these three variables was set in private zone, where anything outside the class cannot access.

3.2.2 Input

There are four ways for input. The first on is input value when declaring the class. Then, the value can be reset at any time use Fraction.set() function. This function accepts both fraction and double type parameters. The third way is use cin stream to input. Just as other type number, you can directly use cin >> Fraction to input a new value for Fraction. This kind of input accept double and also fraction format like this '-3/4'. The last way to input is directly assigning value such as Fraction = 3.141.

3.2.3 Output

there are mainly four output formats. The first one is directly use Fraction in a cout stream like this cout << Fraction. This will push a string to cout in format like '-4/3'. The second way is use Fraction.toStr() function. This will return a string just like it shows in cout stream. The third way is use Fraction.val(). This will return a double decimals. The fourth way is to use Fraction.top() and Fraction.bottom() to get its numerator and denominator.

4. Design

4.1 Int Vector Input Algorithm

To get the int type value from input, we first storage the cin stream to string. Then, for each char in string, we use ASCII to judge if its is a number. If it is a number, we will find the end of the number an put it into the vector. While the char is not a char, we will ignore it and to look at the next char in the string. This process repeats until meet the end of the string.

Since cin automatically deals the '\n', in this scope, we only need to consider if a char is number or not number.

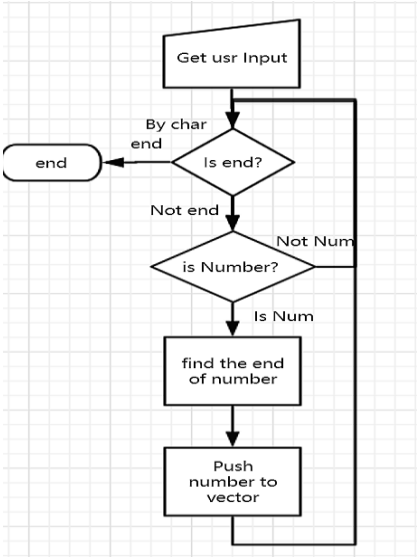

Figure 1 – Int Vector Input Algorithm

4.2 Vector Comparison Algorithm 1

The first algorithm to compare two vectors is as shown in Figure 2. The thought is, for each element in vector A, we find if there is a same element in vector B. And then we record this result. Then, we let A as the former B and B as the former A and repeat this process. Which is to say, for every element in former B, we find if there is a same one in former A. If the forward and reverse comparison both return true, then we can confirm that these two vectors are same.

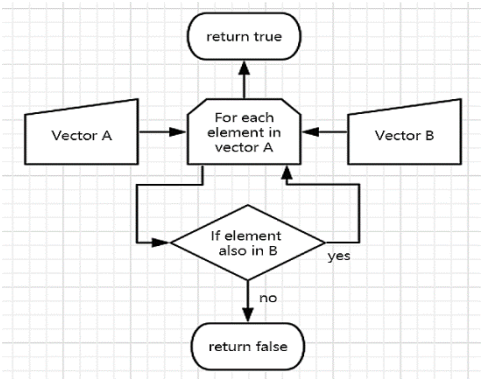

Figure 1 – Vector Comparison Algorithm 1

4.3 Vector Comparison Algorithm 2

The second method is that we sort the element of each vector first use the sort() function from standard library. This function can sort the element in certain order and will not delete the repeated element. Thus, for each vector after being sorted, we find the next different element and compare them. If they are all the same, then the two vectors are same. Otherwise, the two vector will be not the same.

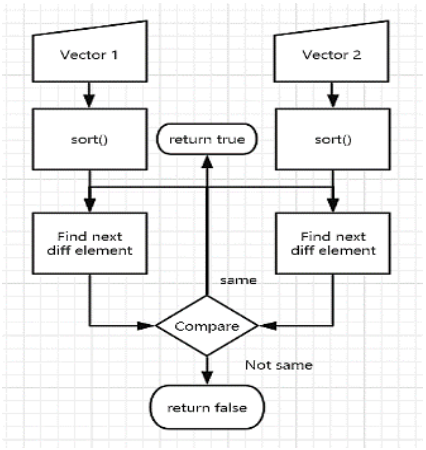

Figure 2 – Vector Comparison Algorithm 2

4.4 Find Common Divisor

There are many methods to find the greatest common divisor of two number. One of the most classical way is to get the remainder of the two number, then replace the two number and let the remainder to replace the bigger number. Repeat this process until the reminder approach zero. Under this condition, the larger number is the greatest common divisor we want. This process is shown in Figure4.

To solve this problem, we include the Multithread Process algorithm as shown in Figure 4. When it is time to insert new data, the program will not stop the whole game to wait for the internet, but try to create a new thread to execute this mission. In this case, when the program uploads the data underground, the game will continue the next round without hesitation.

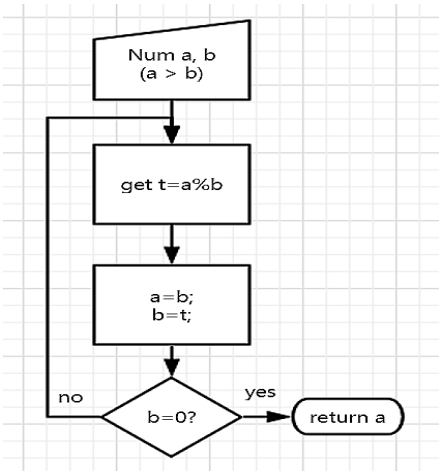

Figure 3 – Find Common Divisor

4.5 Transform Repeated Decimals to Fraction

According to mathematic, every repeated decimal is rational number and can be transform to fraction. Under this circumstance, this algorithm tries to achieve this goal. For a repeated decimal, the most significant part is its repeated part. To get this part, we need to remove the integer part of the double type decimal and remain the decimal part. Then, transfer the decimal part from double to unsigned long int type for the convenience of the following process. Use recursion to find the repeat part. However, the repeat part at this time could not be exactly in the right order. For example, if the repeat part is '234', at this stage the repeat part recognized might be '342' or '423'. Thus the next step is to check if the order of the repeat part is correct.

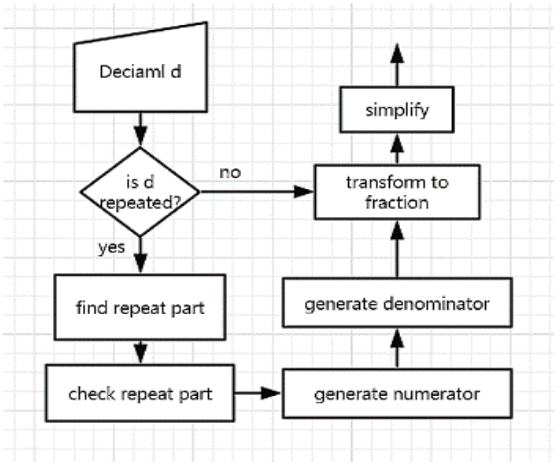

Figure 5 – Transform Repeated Decimals to Fraction

After successfully get the repeat part, the integer part and the left part, we can use mathematic method to compute the numerator and . According to material on internet, the numerator of a repeated decimal is equal to the unrepeated part and first repeated part of the decimal subtract the first repeat part. The denominator part is a combination of the repeat part length's 9 and the not repeat part length's 0. For example, for decimal .4232323, the repeat part is 23, which length is 2. And its non-repeat part is 4, which length is 1. In this case, the denominator would be 990. Our algorithm is a represent of this math method in C++ code format.

5. Implementation

There are four Cpp file of this assessment. Since I have written some codes that belong to C11, it is recommended to use a complier such as gcc. I have prepared a gcc automatic compile script named compilergcc.bat for help.

same_vec.cpp

This file contains all of the source code and test code of the first exercise. You can also get it from the following url.

url: https://github.com/string1995/eee102/blob/master/as1/same_vec.cpp

Fraction.h

This file includes the source code of Fraction Class which belong to the second exercise. It is also support online view from the following url.

url: https://github.com/string1995/eee102/blob/master/as1/Fraction.h

Fraction_test.cpp

This file contains the test code of Fraction Class.

url: https://github.com/string1995/eee102/blob/master/as1/Fraction_test.cpp

6. Test

6.1 Vectors Comparison

Input Test

When input a series of numbers ended by a return key and be separated by not numbers, the program should correctly recognize the number and push them into the vector.

When the input is in normal forsmat.

Test result:

```
Please input Vector A: 1, 2, 3, 4
Please input Vector B: 1, 2, 3, 4,
Vector A: 1, 2, 3, 4,
Vector B: 1, 2, 3, 4,
Result from algorithm 1: Same!
Result from algorithm 2: Same!
```

Figure 6 – Test Result of Input in normal format

When the input is strange and numbers are mixed with chars that difficult to deal with.

Test result:

```
Please input Vector A:   ty34qfja32r0)*KPO23i-4ak[]0i3
Please input Vector B: 349 08h@R@AF2rq=23rf;
Vector A: 34, 32, 0, 23, 4, 0, 3,
Vector B: 349, 8, 2, 23,
Result from algorithm 1: Not Same!
Result from algorithm 2: Not Same!
```

Figure 7 – Test Result of strange input

As you can see from Figure 7, the numbers were recognized even they are mixed with several very strange characters. However, there is one the need to be point out. In the vector A, there is a '-4' in the input context. However, this part was recognized to be '4' instead of '-4'.

I finally decide not to fix this problem since I am a little confused about the requirement in the instruction that "delimiter can be anything other than numbers". As I don't know if '-' is one part of numbers, I decides not to regard this as a bug.

Comparison Test

When elements in these two vectors are same but in different order, they should be regarded as same.

Test result:

```
Please input Vector A: 1,2,3,4
Please input Vector B: 4,3,2,1
Vector A: 1, 2, 3, 4,
Vector B: 4, 3, 2, 1,
Result from algorithm 1: Same!
Result from algorithm 2: Same!
```

Figure 8 – Test Result of Elements in different order

The repeat of element should not influence the judgment.

Test result:

```
Vector A: 1, 2, 2, 3, 4,
Vector B: 1, 2, 3, 3, 4, 4,
Result from algorithm 1: Same!
Result from algorithm 2: Same!
```

Figure 9 – Test Result of repeat elements

Contain relation should not be regarded as same.

Test result:

```
Vector A: 1, 2, 3,
Vector B: 1, 2, 3, 4,
Result from algorithm 1: Not Same!
Result from algorithm 2: Not Same!
```

Figure 10 – Test Result of B contains A

```
Vector A: 1, 2, 3, 4,
Vector B: 1, 2, 3,
Result from algorithm 1: Not Same!
Result from algorithm 2: Not Same!
```

Figure 11 – Test Result of A contains B

6.2 Fraction Class

Input Test

Input when declaration, all input should be normalized and simplified.

Test result:

```
Fraction a;
Fraction b(3,-4);
Fraction c(5);
Fraction d(-3.14);
Fraction e = 1.333333;
Fraction f = -b;
```

```
Output in Fraction Form:
a: 0
b: -3/4
c: 5
d: -157/50
e: 4/3
f: 3/4
```

Figure 12 – Input Test of declaration

Output Test

Output test using cout directly.

Test result:

```
cout << "Output in Fraction Form: " << endl;
cout << "a: " << a << endl;
cout << "b: " << b << endl;
cout << "c: " << c << endl;
cout << "d: " << d << endl;
cout << "e: " << e << endl;
cout << "f: " << f << endl << endl;
```

```
Output in Fraction Form:
a: 0
b: -3/4
c: 5
d: -157/50
e: 4/3
f: 3/4
```

Figure 12 – Output Test of direct cout

Output test of decimal.

Test result:

```
cout << "Output in Decimals Form: " << endl;
cout << "a: " << a.val() << endl;
cout << "b: " << b.val() << endl;
cout << "c: " << c.val() << endl;
cout << "d: " << d.val() << endl;
cout << "e: " << e.val() << endl;
cout << "f: " << f.val() << endl << endl;
```

```
Output in Decimals Form:
a: 0
b: -0.75
c: 5
d: -3.14
e: 1.33333
f: 0.75
```

Figure 13 – Output Test of decimals

Output test of numerator.

Test result:

```
cout << "Output Numerator: " << endl;
cout << "a: " << a.top() << endl;
cout << "b: " << b.top() << endl;
cout << "c: " << c.top() << endl;
cout << "d: " << d.top() << endl;
cout << "e: " << e.top() << endl;
cout << "f: " << f.top() << endl << endl;
```

```
Output Numerator:
a: 0
b: -3
c: 5
d: -157
e: 4
f: 3
```

Figure 14 – Output Test of numerator

Output test of denominator.

Test result:

```
cout << "Output Denominator: " << endl;
cout << "a: " << a.bottom() << endl;
cout << "b: " << b.bottom() << endl;
cout << "c: " << c.bottom() << endl;
cout << "d: " << d.bottom() << endl;
cout << "e: " << e.bottom() << endl;
cout << "f: " << f.bottom() << endl << endl;
```

```
Output Denominator:
a: 1
b: 4
c: 1
d: 50
e: 3
f: 4
```

Figure 15 – Output Test of denominator

Operator Test

Let b = -3/4. Test the opposite number -b and reciprocal ~b.

Test result:

```
cout << "-b = " << -b << endl;
cout << "~b = " << ~b << endl << endl;
```

```
-b = 3/4
~b = -4/3
```

Figure 16 – Operator Test of – and ~

Let b = -3/4, e = 4/3 and c = 5. Test operator +, -, *, / and %.

Test result:

```
cout << b << " + " << e << " = " << b+e << endl;
cout << b << " - " << e << " = " << b-e << endl;
cout << b << " * " << e << " = " << b*e << endl;
cout << b << " / " << e << " = " << b/e << endl;
cout << e << " % " << c << " = " << e%c << endl << endl;
```

```
-3/4 + 4/3 = 7/12
-3/4 - 4/3 = -25/12
-3/4 * 4/3 = -1
-3/4 / 4/3 = -9/16
4/3 % 5 = 4/3
```

Figure 17 – Operator Test of +, -, *, / and %

Let b = -3/4, e = 4/3. Test operator +, -, *, / and % interreact with other types of number.

Test result:

```
cout << b << " + " << 4.44 << " = " << b+4.44 << endl;
cout << 4.44 << " + " << b << " = " << 4.44+b << endl;
cout << b << " - " << 3 << " = " << b-3 << endl;
cout << b << " * " << 2.33333 << " = " << b*2.33333 << endl;
cout << 2.33333 << " * " << b << " = " << 2.33333*b << endl;
cout << b << " / " << 5 << " = " << b/5 << endl;
cout << e << " % " << 1 << " = " << e%1 << endl << endl;
```

```
-3/4 + 4.44 = 369/100
4.44 + -3/4 = 369/100
-3/4 - 3 = -15/4
-3/4 * 2.33333 = -7/4
2.33333 * -3/4 = -7/4
-3/4 / 5 = -3/20
4/3 % 1 = 1/3
```

Figure 18 – Operator Test of +, -, *, / and % with other type

Let b = -3/4. Test b++, ++b, b--, --b.

Test result:

```
cout << "b++ = " << b++;
cout << "; b = " << b << endl;
cout << "++b = " << ++b;
cout << "; b = " << b << endl;

cout << "b-- = " << b--;
cout << "; b = " << b << endl;
cout << "--b = " << --b;
cout << "; b = " << b << endl << endl;
```

```
b++ = -3/4; b = 1/4
++b = 5/4; b = 5/4
b-- = 5/4; b = 1/4
--b = -3/4; b = -3/4
```

Figure 19 – Operator Test of b++, ++b, b--, --b

Let b = -3/4 and c = 5. Test assignment +=, -=, *=, /=.

Test result:

```
cout << "b += c; b = " << (b+=c) << endl;
cout << "b -= c; b = " << (b-=c) << endl;
cout << "b *= c; b = " << (b*=c) << endl;
cout << "b /= c; b = " << (b/=c) << endl << endl;
```

```
b += c; b = 17/4
b -= c; b = -3/4
b *= c; b = -15/4
b /= c; b = -3/4
```

Figure 20 – Assignment Test of +=, -=, *=, /=

Comparison Test

Let b = -3/4, c = 5 and f = 3/4. Test >, <, >=, <=, ==, !=.

Test result:

```
cout << "if b > c ?? " << ((b>c)? "Yes" : "No") << endl;
cout << "if b < c ?? " << ((b<c)? "Yes" : "No") << endl;
cout << "if b >= c ?? " << ((b>=c)? "Yes" : "No") << endl;
cout << "if b <= c ?? " << ((b<=c)? "Yes" : "No") << endl;
cout << "if b == c ?? " << ((b==c)? "Yes" : "No") << endl;
cout << "if b == -f ?? " << ((b==-f)? "Yes" : "No") << endl;
cout << "if b != c ?? " << ((b!=c)? "Yes" : "No") << endl << endl;
```

```
if b > c ?? No
if b < c ?? Yes
if b >= c ?? No
if b <= c ?? Yes
if b == c ?? No
if b == -f ?? Yes
if b != c ?? Yes
```

Figure 21 – Comparison Test of >, <, >=, <=, ==, !=

Divide 0 Test

Declare a new Fraction with denominator equals 0.

Test result:

```
try{
    Fraction g(1,0);
}catch(const char* msg){
    cout << msg << endl;
}
```

```
Division by zero condition!
```

Figure 22 – Divide 0 Test

cin Input Test

Input with cin in fraction format.

Test result:

```
Please Input a Fraction(-3/5) or a Decimals(e.g. 3.14): -30/24
-5/4
```

Figure 23 – cin Input Test in Fraction

Input with cin in decimal fromat.

Test result:

```
Please Input a Fraction(-3/5) or a Decimals(e.g. 3.14): 5.28
132/25
```

Figure 24 – cin Input Test in Decimal

7. Conclusion

In this report, we have deeply explored how two int type vectors can be compared and how to build a practical fraction class which can almost be treated just as the other number type in C++. By finishing this assignment, I got a deeper understanding concerning object-oriented programing and also had a general and useful perspective on C++ class. Different from the struct in C, class seems to be more flexible and easier to practice. With classes, it tends to be more enjoyable to program since there are more room in program that you can define by yourself, instead of just following the standard library. This feature is also very exciting when cooperating with others. As every function can be classified into classes, and each class can be designed to be a black box which users can use it without knowing what exactly going on in the box. This makes collaboration easier to farm-out and developed.

After this report, I deeply feel that there are reasons that to develop C++ even that C had seemed to be enough powerful. However, there are still shortage of C++. For instance, it does not have a unified library and package control system. This make C++ a little embarrassed compared with other language such as js and go, while C++ has its extraordinary perform in execute efficiency.