

EEE102 C++ Programming and Software Engineering II

Assessment 2

Report

Prepared by: Xuhui Gong

ID Number: 1718128

April 6, 2019



Xi'an Jiaotong-Liverpool University

西交利物浦大學

Abstract:

This report aims to explain the software development process (SDP) in the two main programs which are vectors comparison and fraction manipulation program. For the first step of SDP is problems specification which focus on the basic problems. Then, for second step is analysis, it focus on the how to write a function to solve the iFraction. Thirdly, it is the most difficult part which is algorithms design which needs to develop a list of steps to solve the problem. Using the algorithms to achieve on code is the fourth step. Finally, testing and verification is also important part, bugs will be found in this part, then the program writer will be fix the bugs.

Contents:

1. Introduction.....	4
2. Problems Specification	4
3. Analysis the Problems.....	6
4. Algorithms Design.....	9
5. Implementation.....	26
6. Testing and Verification.....	27
7. Conclusion.....	37

1. Introduction

The first exercise aims to derive a sub-class which from the base class Fraction in the last assessment. By using the SOP, this report will provides a complete step to solve the problem. Then second exercise is an uncomplicated program, which needs to full the blank and add other two characters. However, the balance of the game is also reflected in this report and the above operations will also follow the SOP principles.

2. Problem Specification

2.1 Exercise 1

The first exercise needs to write a sub-class iFraction which based on the Fraction. It is worth noting that the program needs to rewrite the operator such as addition, subtraction, multiplication and division. The advantage of this is that you can directly extract the addition in the operation instead of the original *FracA.add(FracB)*. In addition, we also need to inherit the Fraction class. This is

the only easy way to complete a program that shows the true score. The main difficulty of this job is how to inherit and rewrite the method. And the ConvertF method should be written in the way of a friend function. Its role is mainly to facilitate access to the variables of the class without the need to access and rewrite through methods.

2.2 Exercise 2

The second exercise needs to finish a Role-playing game. First of all, students should complete the question mark part and add the necessary header file to make the file run normally. To understand the game, you need to understand the operating mechanism of the file first. There are three different roles in this game, one is swordsman, the other is archer, and the other is mage. In the running of the game, players can choose these three roles, and robots are random roles in these three roles. Players will face multiple games against robots. In the course of the game, players can use healing potions and magic potions to increase their blood and magic power respectively. There is a special class in the program to control the backpack of players and robots which is called container. Relationships based on other classes are discussed below.

3. Analysis the Problems

3.1 Exercise 1

For exercise one, the fraction class has **two private variables** which are top and bottom, so the input should let users input at least two fraction that they could do other manipulation such as add. So in this program, it introduces two input variables which are:

int top_first

int bottom_first

By using one function, this program repeatedly let user input and store the user input and wait it to process. So this program uses a **few functions** to distinguish digital or others correctly.

Because it has few functions in this program, this program introduces **two intermediate variables** which are:

Fraction first_fraction;

Fraction second_fraction;

For the **output** of exercise two, the program should give the result of manipulation, in this program, **it returns a fraction class** which passes to the result in main document:

Fraction result;

In some situations, **error input could cause system breakout**, by using some statement such as *try()*, the program could avoid this thing happened and **give the correct feedback**. Other output is fraction should be converted to the decimal and decimal converts to fractions.

Finally, Fraction should be converted to the iFraction, so **the convertF should have parameter of Fraction, then return a iFraction**. However, this function should be set as a **friend function**. Then, the proper fraction will be shown in the screen like $5\frac{1}{2}$, which will be shown 5 and $\frac{1}{2}$ in the screen.

3.2 Exercise 2

For exercise two, the program should let user to input the **character's name**, then for the battle, user should **input the number related to the selection**. However, At the end of each battle, if the player wins, **player will get the booty of the computer, such as blood tonic and magic potion**. This is also an input operation, because each time the game is settled, the amount of drugs in the computer will be searched and the total amount of drugs available to the player will be added. In addition, **battle information** such as player selection and computer selection will be displayed on the screen in the form of information, such as **player and computer level, player and computer role, player and computer experience value, and player's remaining drug number will be displayed**. And the **result of each battle** will be seen as an output by the player. Finally, the necessary error display is also displayed on the screen in the form of output. For example, when the player enters the option incorrectly, the program will let the player re-enter.

4. Algorithm Design

4.1 Exercise 1

For exercise one, this report will discuss in three parts which are the **relationship of class and sub-class, operator overloading, application of Friend Function and fraction's input and Calculate. The following algorithms have been tested to ensure availability.**

4.1.1 Relationship of class and sub-class

The class which is named Fraction is a base class, so the sub-class is iFraction.

iFraction inherits all the features of Fraction. When the iFraction class is declared, the following syntax is used:

public Fraction

However, this public could be changed by private or protected, using the above statement can impose some restrictions on class access. If we use the keyword public, we can directly access the same class and derived classes as well as external classes. If the keyword protected is used, external classes cannot be accessed. If the keyword private is used, access to the class is limited to the class

itself. Because this program does not need too many restrictions, the keyword `public` is used here. The relationship between `Fraction` and `iFraction` are shown in the Figure 1. Besides, it has its own fields: *whole_number*, it means that the real value of the fraction, eg. $5\frac{1}{2}$'s 5.

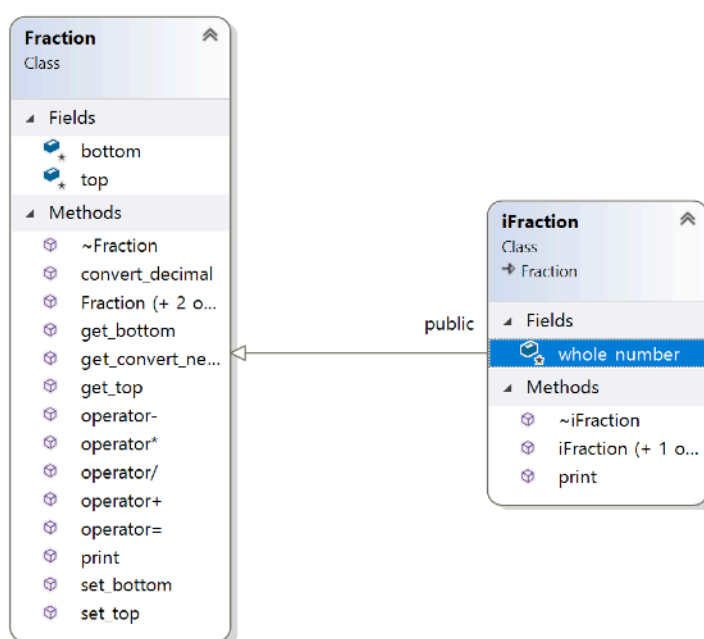


Figure 1: The relationship between `Fraction` and `iFraction`

4.1.2 Operator overloading

The operator like add, subtract, multiply and divide in the initial procedure is not very intuitive. For example: If the program generates an instance object: *Frac1* and *Frac2*. So if the program want to calculate the sum of *Frac1* and *Frac2*. In the original procedure, it should be written as follows:

Fraction result = Frac1.add(Frac2)

When the program modifies the operator, the calculation process looks more straightforward.

Fraction result = Frac1+Frac2

The method of overloading operators is simple.

Fraction operator + (Fraction Frac);

Fraction operator - (Fraction Frac);

Fraction operator * (Fraction Frac);

Fraction operator / (Fraction Frac);

Through the above definition, the program can directly identify the operator and calculate the result according to the definition of the previous job.

4.1.3 Friend Function

By binding a friend function on the class, the variables of the class can be accessed directly within the function without the need for getter and setter methods. In this program, friend functions are directly bound to Fraction and iFraction, which means that the function can directly access the values in these two classes. Finally, the realization of friend function is carried out under the main function file. The schematic diagram is shown in the Figure 2.

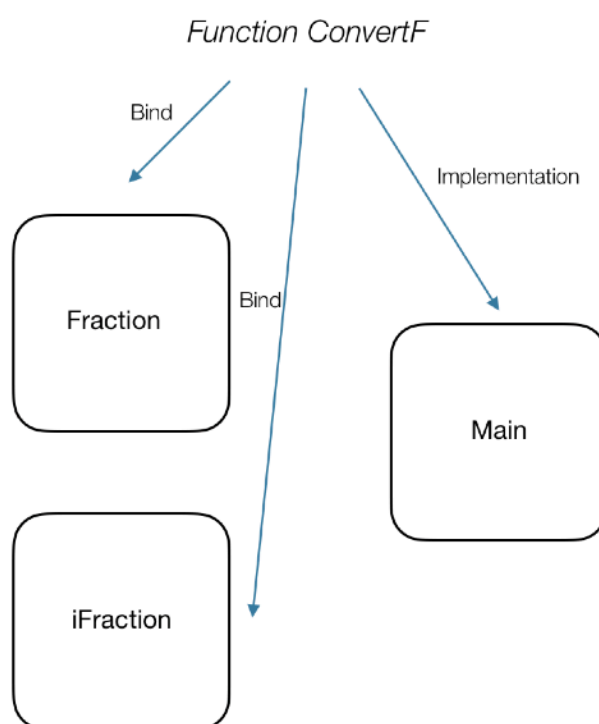


Figure 2: Schematic diagram of ConvertF function

4.1.3 Fraction's Input and Calculate

The algorithm of adding method is considering top and bottom, if the program knows that two fraction and using getter, the main program could get each fraction's value, then using adding method to process. It will show the basic principle of adding two fractions. Consider two fractions: $\frac{3}{5}$ and $\frac{10}{3}$

For denominator, the value is equal to the product of the denominators of the two fractions.

*Denominator's result: $5 * 3$*

This means:

*Denominator's result = Fraction_one's denominator's result * Fraction_two's denominator's result*

This is the same as the Numerator

Writing to the program:

```
int bottom_result = bottom * bottom_two;
```

```
//get the bottom of result
```

```
int top_result = top * bottom_two + top_two * bottom;
```

//get the top of result

This program has some limitations to input, because let users to input each fraction's top and bottom step by step it is very easy to control the program. However, if user input are not a number, the program will identify the user input and give some feedback such as invalid input. It is very similarly to the first assessment's input function. **To avoid some errors in the program, this program has the function named *bool isinput_all_digital(string user_input)*. Then, the input molecules and denominators will be stored as fractional types. Finally, through the friend function `ConvertF`, the program converts `Fraction` into `iFraction` type. This process is shown in Figure 3.**

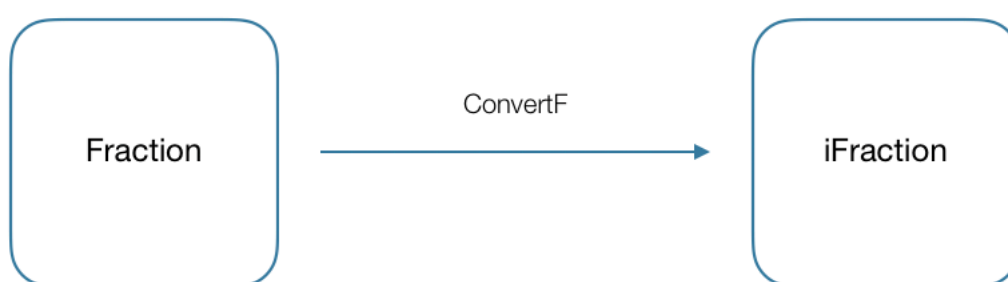


Figure 3: Processing procedure

4.1.3 ConvertF Algorithm

Because the iFraction has been create for a constructor, in the ConvertF function, the program only needs to convert Fraction to the iFraction. The flow chart is shown in the Figure 4.

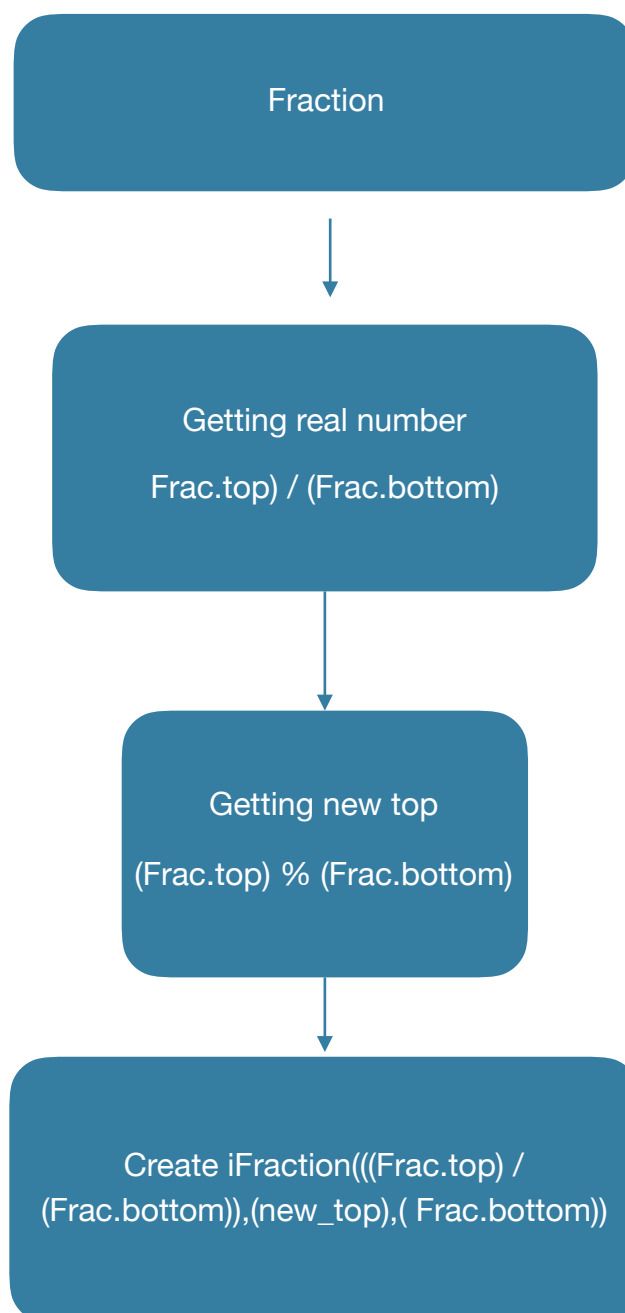


Figure 4: Flow chart of ConvertF

However, this may cause errors when the user enters - 4/1 add - 2/1. At this point, the molecule becomes zero. So the program later added if statements to prevent errors.

4.2 Exercise 2

This report will focus on. header files, because they are declarations of functions. It will describe container. h and player. h, swordsman. h and main respectively. In addition, the CPP files of persons and their CRC CARD will be described.

4.2.1 Container.h

First of all, the header file can see the declaration of the file, for the backpack, it should have the players and enemies of blood and magic potions. This class sets a default structure to determine the default values of the two potions.

```
container::container()
{
    set(0, 0);
}
```


Then the procedure declared the setter and getter the number of heal and the number of magic waters. Also, if the player or computer wants to use the heal or magic water, the method of useHeal and useMW is written in this program. However, the player need to see the number of magic water and heals, so the method which is named display shows the information to the player. The UML and CRC Card is shown in the Figure 5 and Figure 6.

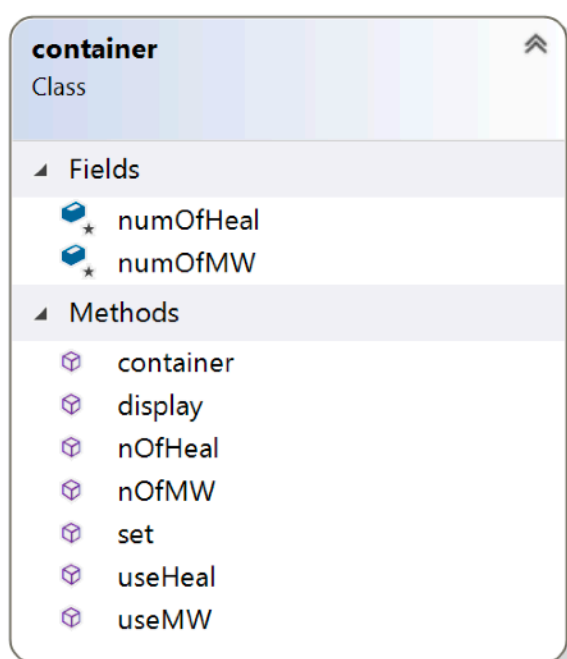


Figure 5: UML of Container

Class: Container	
Responsibilities	Collaborators
Provides a Container class for Player	Player
Make the instance object of Player have Container by default	
Two items, blood bag and magic potion, were declared.	

Figure 6: CRC Card of Container

4.2.1 Player.h

For this class, the program designed four friend functions to facilitate access and modification of variables in player, rather than using setter and getter. In fact, this solution is a compromise between packaging and convenience.

friend void showinfo(player &p1, player &p2);

friend class swordsman;

friend class magician;

friend class archer;

This class uses the enumeration method to list three different occupations. Finally, the classes of these three occupations inherit this class. And this class defines

variables such as blood volume, mana value, DP, experience value, rank, wisdom, luck, but sets these variables as protected. However, since the program has already set the inherited classes as friend classes, the classes represented by the three roles can access Player's values directly. In addition, the methods that roles can use are already set up in Player, and role classes can directly access the open methods in Player classes. For example, blood return, etc. Finally, the player's decision of death only needs to be defined in Player. This method is set to Private, because this method does not need to be inherited, but only as a judgment. The CRC Card and UML has shown in the Figure 7 and Figure 8.

Class: Player	
Responsibilities	Collaborators
The Player class provides the basis for three role classes	Container
Collaboration in the main function	Swordsman
Provides the method of setting and acquiring basic information of players.	Archer
	Mage

Figure 7: CRC Card for Player

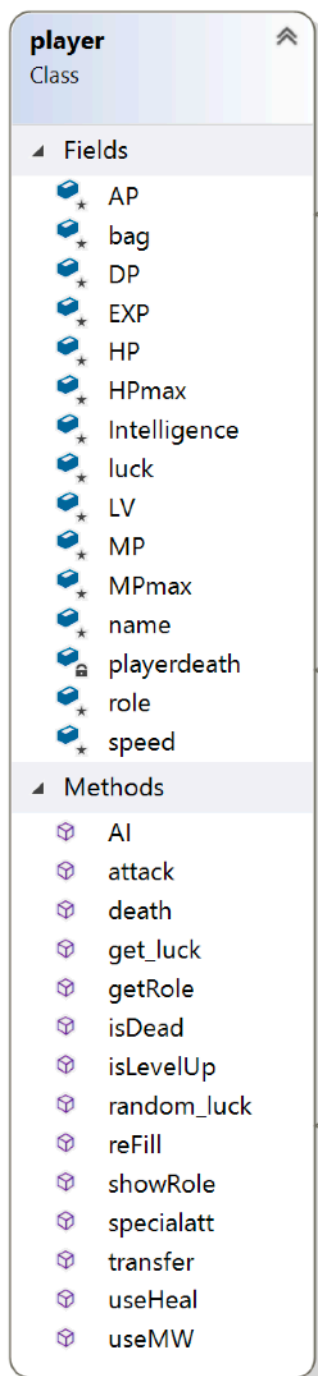


Figure 8: UML for Player

4.2.2 Swordsman.h, Archer.h and Mage.h

These classes are subclasses of Player. In fact, considering the later development of the game, more characters can be added to the program, which can use the same mode, but the specific attack ways of different characters may be different. When inheriting fathers, it should be noted that the construction method cannot be inherited, so each new character needs its own construction method. In the Base class, polymorphism is used and only declared in the Base class. The concrete implementation is implemented in the subclass. Therefore, attacks and other methods are implemented through subclasses. Their UML and CRC CARD are shown below in the Figure 9 and Figure 10.

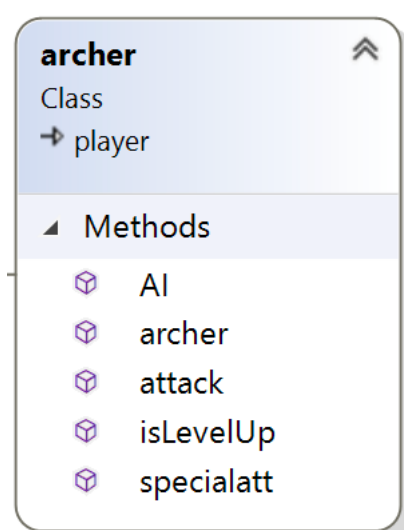


Figure 9: Archer's UML(Only shows Archer because other characters are the same)

Class: Archer	
Responsibilities	Collaborators
Interaction in the main function	Player
The Final Embodiment of Base Class	Swordsman
Providing methods of attack	Mage
Experience upgrade system	

Figure 10: Archer's CRC Card

4.2.3 Main.CPP

In the main function, it is very important to use all kinds of classes correctly.

The main function first declares a null pointer about Player, which is not a good habit when writing C++ in normal time. Of course, because the new version of the compiler can recognize this error, but declaring null pointer directly without pointing to any variable may lead to program errors. However, the reason why a null pointer is declared is that the program is not sure about a new type to be created in the process of writing. If it is simply declared in the if conditional

sentence, the readable range of the new type is only in brackets, so using pointers is a better solution to this problem.

*player *human;*

Here is a code that creates enemies and selects them by random numbers.

```
player *enemy;
enemy = new swordsman();//default setting
srand(int(time(0)));//Random Number Seed
if ((rand() % 3 + 1) == 1)
{
    enemy = new swordsman (i, "Junior");      // Initialise an opponent, level
i, name "Junior"
}
else if ((rand() % 3 + 1) == 2)
{
    enemy = new archer(i, "Junior");      // Initialise an opponent, level i,
name "Warrior"
}
else
{
    enemy = new magician(i, "Landa"); // Initialise an opponent, level i,
name "Landa"
}
```

Furthermore, a luck variable is defined in the Player class, which is reflected in the main function. The definition is: when a player attacks, he has a 30% chance of attacking the enemy again. This adjustment will help the game continue.

4.2.4 Implementation of Role Class

Since the files have been given in advance, we can make changes in the balance of the game and the way the characters attack. For the swordsman, the main modification of this program is to combine the size of the attack with the character's attack and defense. Considering the attributes of the swordsman's attack, his ordinary attack power is higher than that of other characters. For archer, its special attack and normal attack are balanced. For mages, normal attack is relatively low, but special attack is very high. **And in order to distinguish the attacking ways of three characters, this program adds the method of identifying characters and adds it to the main function. The final result is that the special attack names of three different characters in the main function are different.**

Here's a code example:

```
if (human->getRole() == sw)
    {
        cout << "Chop ";
    }
else if (human->getRole() == mg)
    {
        cout << "Fire ball ";
    }
else
    {
        cout << "Shoot ";
    }
```


4.2.5 Summary of Inheritance Relations

The following is a summary of the inheritance relationship:

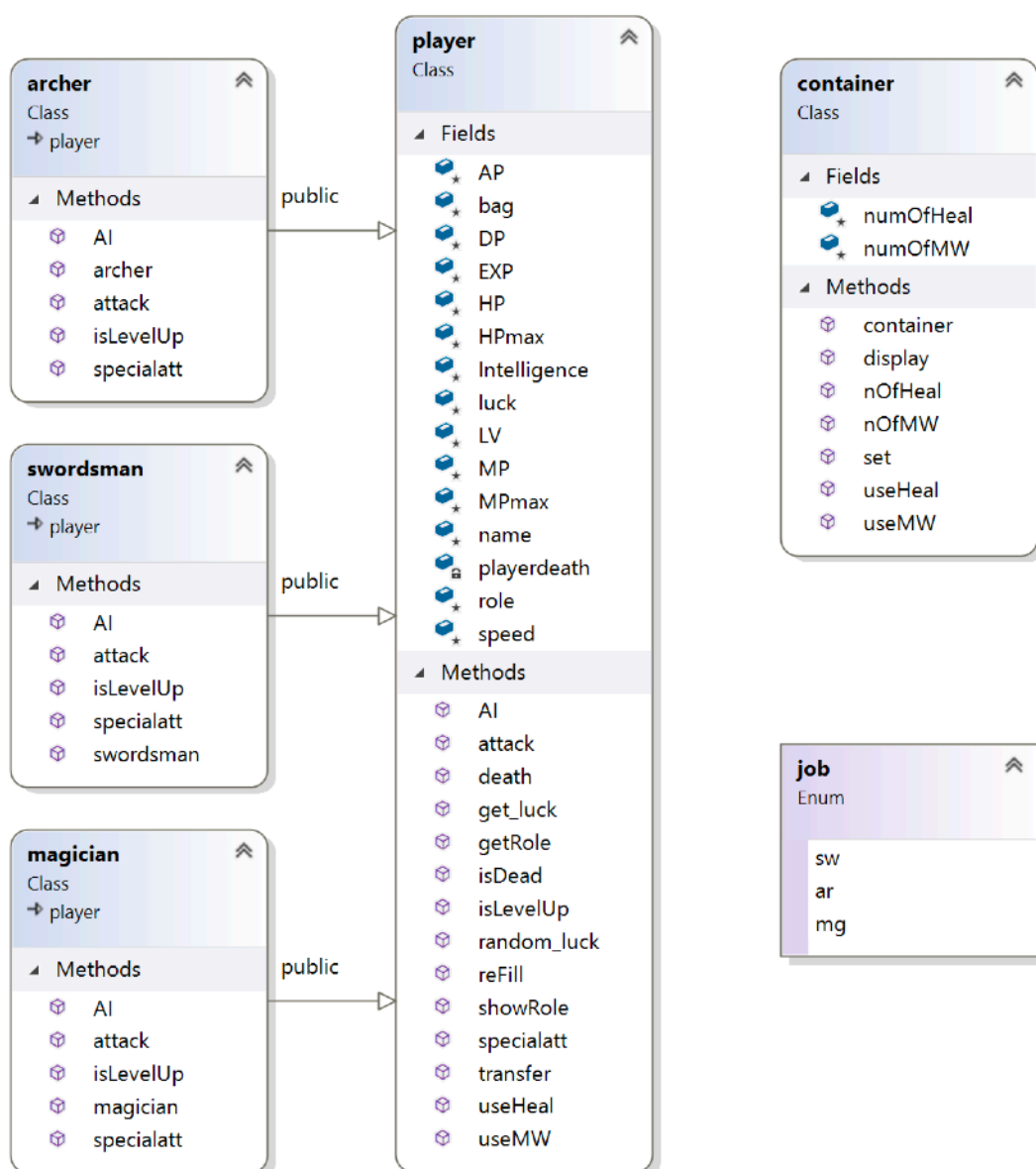


Figure 11: Summary of Inheritance Relations

5. Implementation

- *The first algorithm of the first part see as Assignment 1*
- *The second algorithm of the first part see as Assignment 1_1*
- *The second algorithm of the first part: see as Assignment 1_2*

Note: For Visual Studio, click .solution it is easy to run

6. Testing and Verification

6.1 Exercise 1

This testing only testing the function of `ConvertF`, it is because other content has been tested in the last assignment.

First, test general variables in the Figure 12.

```
Please set up first fraction
Firstly, please enter the top of fraction
5
Secondly, please enter the bottom of fraction
2
*****
Please set up second fraction
Firstly, please enter the top of fraction
4
Secondly, please enter the bottom of fraction
3
Please select which method you want to use
1.Add 2.Subtract 3.Multiple 4.Divide 5.Comparison of size
*****
1
The result is 3 and 5/6
```

```
Please set up first fraction
Firstly, please enter the top of fraction
5
Secondly, please enter the bottom of fraction
1
*****
Please set up second fraction
Firstly, please enter the top of fraction
2
Secondly, please enter the bottom of fraction
5
Please select which method you want to use
1.Add 2.Subtract 3.Multiple 4.Divide 5.Comparison of size
*****
2
The result is 4 and 3/5
```

```
Please set up first fraction
Firstly, please enter the top of fraction
4
Secondly, please enter the bottom of fraction
3
*****
Please set up second fraction
Firstly, please enter the top of fraction
2
Secondly, please enter the bottom of fraction
3
Please select which method you want to use
1.Add 2.Subtract 3.Multiple 4.Divide 5.Comparison of size
*****
3
The result is 8/9
Convert to decimals, the result is 0.888889
*****
```

```
Please set up first fraction
Firstly, please enter the top of fraction
3
Secondly, please enter the bottom of fraction
4
*****
Please set up second fraction
Firstly, please enter the top of fraction
5
Secondly, please enter the bottom of fraction
13
Please select which method you want to use
1.Add 2.Subtract 3.Multiple 4.Divide 5.Comparison of size
*****
4
The result is 1 and 19/20
Convert to decimals, the result is 1.95
```

Figure 12: Normal Testing

Here, all the values of the addition, subtraction, multiplication and division operations are tested, no problem.

However, when special values are entered, the program may be affected, such as $4/-1+2/-2$.

In the test of the first generation program, the molecule is zero. This is because the method of calculating residuals is directly used in the iFraction re-conversion. When the scores just entered, the result will be wrong. The error results are as follows:

```
Please set up first fraction
Firstly, please enter the top of fraction
4
Secondly, please enter the bottom of fraction
-1
*****
Please set up second fraction
Firstly, please enter the top of fraction
5
Secondly, please enter the bottom of fraction
1
Please select which method you want to use
1.Add 2.Subtract 3.Multiple 4.Divide 5.Comparison of size
*****
1
The result is 1 and 0/1
Convert to decimals. the result is 1
```

Figure 13: Error output

After further modification, the program can read out special cases, and the problem is fixed in the second version. Test as shown in the Figure 14.

```
Please set up first fraction
Firstly, please enter the top of fraction
4
Secondly, please enter the bottom of fraction
-1
*****
Please set up second fraction
Firstly, please enter the top of fraction
5
Secondly, please enter the bottom of fraction
1
Please select which method you want to use
1.Add 2.Subtract 3.Multiple 4.Divide 5.Comparison of size
*****
1
The result is 1 and 1/1
Convert to decimals, the result is 1
```

Figure 14: Right Testing

6.2 Exercise 2

Now to test the second program, the first thing to test is the program's character selection function. First of all, we chose the Archer role, and the program's character selection was normal. The Figure 15 has shown the testing.

```
#####
# Player   Charlie   LV.   1 # Opponent   Junior   LV.   1 #
# HP 150/150 | MP 75/ 75      # HP 150/150 | MP 75/ 75      #
# AP 25 | DP 25 | speed 25 # AP 25 | DP 25 | speed 25 #
# EXP      75 Job: Archer    # EXP      75 Job: Swordsman    #
-----
Your bag contains:
Heal(HP+100): 1
Magic Water (MP+80): 1
#####
Please give command:
1 Normal Attack 2 Shoot 3 Use Heal; 4 Use Magic Water; 0 Exit Game
```

Figure 15: Character testing

The Figure 16 has shown the other characters testing.

```
#####
# Player   Charlie   LV.   1 # Opponent   Junior   LV.   1 #
# HP 150/150 | MP 75/ 75      # HP 150/150 | MP 75/ 75      #
# AP 25 | DP 25 | speed 25 # AP 25 | DP 25 | speed 25 #
# EXP      75 Job: Mage      # EXP      75 Job: Swordsman    #
-----
Your bag contains:
Heal(HP+100): 1
Magic Water (MP+80): 1
#####
Please give command:
1 Normal Attack 2 Fire ball 3 Use Heal; 4 Use Magic Water; 0 Exit Game
■
```

Figure 16: Other Characters testing


```
#####
# Player   Charlie   LV.   1 # Opponent   Landa   LV.   1 #
# HP 150/150 | MP 75/ 75      # HP 150/150 | MP 75/ 75      #
# AP 25 | DP 25 | speed 25 # AP 25 | DP 25 | speed 25 #
# EXP      75 Job: Swordsman  # EXP      75 Job:   Mage   #
-----
Your bag contains:
Heal(HP+100): 1
Magic Water (MP+80): 1
#####
Please give command:
1 Normal Attack 2 Chop 3 Use Heal; 4 Use Magic Water; 0 Exit Game
```

This program can create enemies of random characters, so this function also needs to be tested. But other problems were found. It is shown in the Figure 17.

```
STAGE1
Your opponent, a Level 1 Swordsman
'\\mac\Home\Desktop\Assignment2_1\Assignment2_1'
CMD.EXE was started with the above path as the current directory.
UNC paths are not supported.  Defaulting to Windows directory.
Press any key to continue . . .
```

Figure 17: Error of testing

This problem is due to the virtual machine I run, so the file path is not the standard Windows path, so after modifying the file path, this problem was fixed.

Next, the game's attack and backpack usage will be tested. Among them, this program adds Luck attributes, players in the ordinary attack, a certain probability can attack again, this content is also tested. It is shown in the Figure 18.

```
#####
# Player   Charlie   LV.   1 # Opponent   Junior   LV.   1 #
# HP 150/150 | MP 75/ 75      # HP 150/150 | MP 75/ 75      #
# AP 25 | DP 25 | speed 25 # AP 25 | DP 25 | speed 25 #
# EXP      75 Job:   Mage   # EXP      75 Job:  Archer   #
-----
Your bag contains:
Heal(HP+100): 1
Magic Water (MP+80): 1
#####
The intelligence of Magician is 25
#####
Please give command:
1 Normal Attack 2 Fire ball 3 Use Heal; 4 Use Magic Water; 0 Exit Game
1
Charlie uses bash, Junior's HP decreases 7
Charlie obtained 8 experience.
Press any key to continue . . .
```

```
Junior's leap attack has been evaded by Charlie
Press any key to continue . . .
```

```
#####
# Player   Charlie   LV.   2   # Opponent   Junior   LV.   1   #
# HP  10/158 | MP  75/ 77   # HP 104/150 | MP  35/ 75   #
# AP  29 | DP  29 | speed  27 # AP  25 | DP  25 | speed  25   #
# EXP    243 Job:    Mage   # EXP    253 Job:   Archer   #
-----
Your bag contains:
Heal(HP+100): 1
Magic Water (MP+80): 1
#####
The intelligence of Magician is 27
#####
Please give command:
1 Normal Attack 2 Fire ball 3 Use Heal; 4 Use Magic Water; 0 Exit Game
1
Charlie uses bash, Junior's HP decreases 9
Charlie obtained 10 experience.
Press any key to continue . . .
Critical attack: Junior uses bash, Charlie's HP decreases 8
Junior obtained 9 experience.
Press any key to continue . . . █
```

Figure 18: Normal attack and Critical attack testing

Lucky mode was also tested. It is shown in the Figure 19.

```
#####
# Player   Charlie   LV.   3   # Opponent   Landa   LV.   3   #
# HP  82/166 | MP  79/ 79   # HP 135/166 | MP  79/ 79   #
# AP  33 | DP  33 | speed  29 # AP  33 | DP  33 | speed  29   #
# EXP    412 Job:    Mage   # EXP    797 Job:    Mage   #
-----
Your bag contains:
Heal(HP+100): 0
Magic Water (MP+80): 1
#####
The intelligence of Magician is 29
#####
Please give command:
1 Normal Attack 2 Fire ball 3 Use Heal; 4 Use Magic Water; 0 Exit Game
1
Charlie uses bash, Landa's HP decreases 13
Charlie obtained 15 experience.
Press any key to continue . . .
Congratulation, you can attack twice
Charlie uses bash, Landa's HP decreases 13
Charlie obtained 15 experience.
Press any key to continue . . . █
```

Figure 19: Luck mode Testing

There are three ways to end the game. The first way is to exit directly, the second way is to be defeated by the enemy, and the third way is to win directly. The following exit will be tested. There are shown in the Figure 20.

```

                                GAME OVER

C:\Game\Assignment2_1\Debug\Assignment2_1.exe (process 7596) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

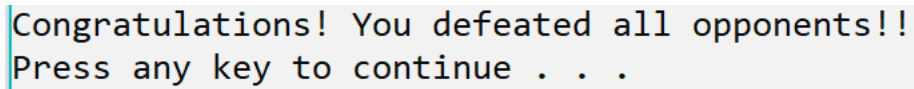
```

```

#####
# Player          C   LV.   1 # Opponent   Junior   LV.   1 #
# HP 150/150 | MP  75/ 75      # HP 150/150 | MP  75/ 75      #
# AP  25 | DP  25 | speed  25 # AP  25 | DP  25 | speed  25 #
# EXP      75 Job: Swordsman  # EXP      75 Job: Archer    #
-----
Your bag contains:
Heal(HP+100): 1
Magic Water (MP+80): 1
#####
Please give command:
1 Normal Attack 2 Chop 3 Use Heal; 4 Use Magic Water; 0 Exit Game
0
Are you sure to exit? Y/N
Y

C:\Game\Assignment2_1\Debug\Assignment2_1.exe (process 6496) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```



```
Congratulations! You defeated all opponents!!  
Press any key to continue . . .
```

Figure 20: Three Types of exiting

7. Conclusion

This report fully describes all the steps that a program generates, whether it is from the beginning of the program, to the consideration of variables, to the choice of algorithms, and the writing of the program, to the final modification of the bugs.