

## Department of Electrical and Electronic Engineering

EEE102 C++ Programming and Software Engineering		
Assignment4 Inheritance and Polymorphism <sup>1</sup> SDP Report		
	Name:Zheng Sun	

I certify that I have read and understood the University's *Policy for dealing with Plagiarism, Collusion and the Fabrication of Data* With reference to this policy, I certify that:

ID Number :\_\_\_\_1507820\_\_\_\_\_

- My work does not contain any instances of plagiarism and/or collusion.
- My work does not contain any fabricated data.

By handing in my assignment for marking, I formally declare that all the above information is true to the best of my knowledge and belief.

Signatu	re: Zheng Sun
_	_
<sup>1</sup> Last updated on April 18, 2017	

#### 1. Problem Statement:

This task aims to implement concepts of class Inheritance and dynamic polymorphism and to reinforce the understanding of OOP by practice code reuse. Overall, the object of the assignment is clear and lucid.

- a) Fill in the seven incomplete code gaps to make the code able to be compiled and run by implement logic deduction with knowledge of inheritance.
- **b)** Develop two new role classes (archer and mage) with attributes different from each other inherited from class player. Introduce attribute luck.
- **c)** Modify the main code to deal with <u>erroneous input</u> and <u>new attributes</u> of the role and the combat to make the RPG both more robust and playful.

#### 2. Analysis:

# a) Inputs:

- 1. String type as player's name. → std::geline(cin, str)
- 2. Int type as job and command selection → int intParser() by@myself
- 3. Char type when exiting the game → char charParser() by@myself

#### b) Output:

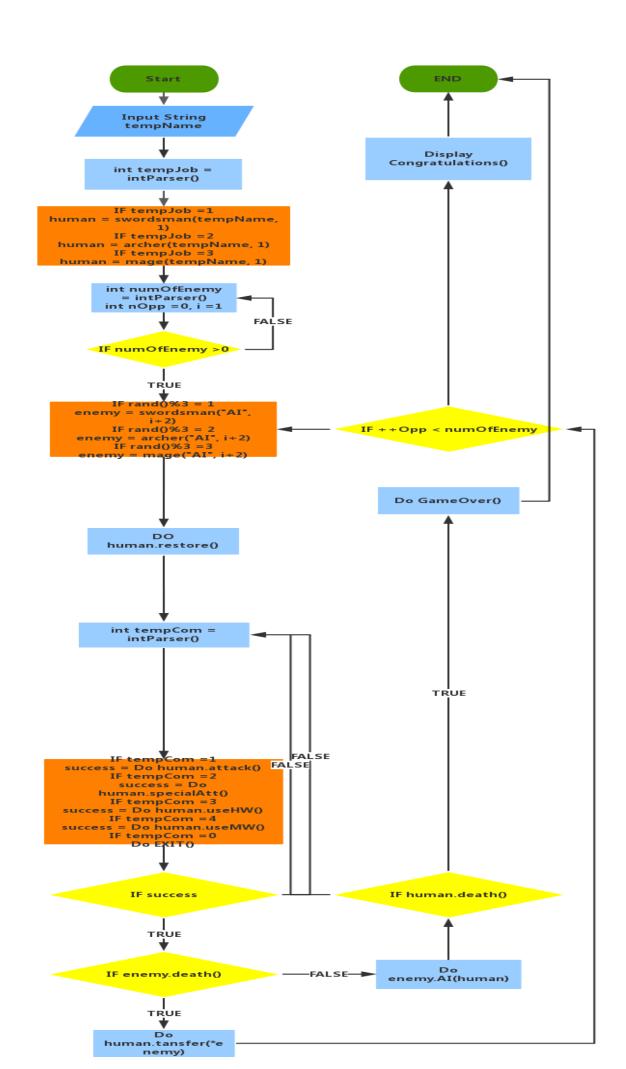
- 1. Interactive instructions prompt user to input name, select job, give orders.
- 2. Informative notice show: current status of the player and opponents to help make decision, results of each combat turn, and LV up information.

#### c) Extra and bonus characteristics

- 1. Attribute **luck** added to calculate the evasion rate and critical attack rate.
- 2. Generally, swordsman →archer → mage → swordsman
- 3. swordsman: high growth in AP & DP & HP, low in MP, speed, luck
  - archer: high growth in speed & luck → highly evasive
  - mage: high growth in intelligence → ignore DP
    - high critical attack factor → sometimes one shoot to kill
- 4. Display different names of normal & special attack of different jobs
- 5. Choose the number of opponents.
- 6. Change mistake info to the correct one, e.g. Magic water increase MP by 100, not 80 as originally presented.
- 7. Many other false or unreasonable arrangements changed, it is recommended to play the game and look at the code to draw conclusion.

## 3. Design

a) main() function design



# b) Class Design: 1.CRC cards and 2.hierarchical UML class graph 1. CRC cards of classes

Class: container		
Responsibility:	Collaborators:	
Hold information of inventory and		
provide interface for player to access	player	
and manipulate inventory.		
Display contents of the inventory.		

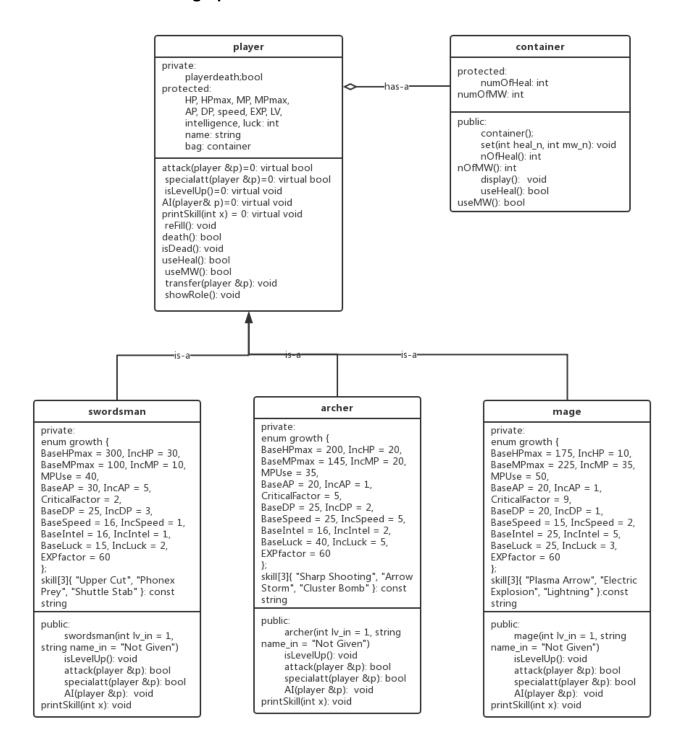
Class: player			
Responsibility:	Collaborators:		
Provide dynamic polymorphism for	swordsman		
the implementation of 3 subclasses.	archer		
Hold basic common data members	mage		
and role behaviours for inheritance.			
Provide interface for container class	container		
manipulation.			
Display full role info and result of			
every turn of the combats.			

Class: swordsman			
Responsibility:	Collaborators:		
Hold specific initial and growth data			
members and behaviours of the role	player		
swordsman.			
Collaborating with inherited base			
class player, conduct combats of			
every turn.			

Class: archer		
Responsibility:	Collaborators:	
Hold specific initial and growth data		
members and behaviours of the role	player	
archer.		
Collaborating with inherited base		
class player, conduct combats of		
every turn.		

Class: mage		
Responsibility:	Collaborators:	
Hold specific initial and growth data		
members and behaviours of the role	player	
mage.		
Collaborating with inherited base		
class player, conduct combats of		
every turn.		

#### 2. Hierarchical UML class graph



#### 4. Implementation:

EEE102\_Assessment\_4\Task2\container.h

EEE102\_Assessment\_4\Task2\container.cpp

EEE102\_Assessment\_4\Task2\player.h

EEE102\_Assessment\_4\Task2\player.cpp

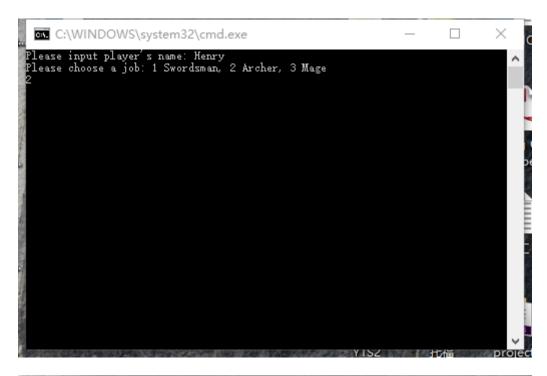
EEE102 Assessment 4\Task2\swordsman.h

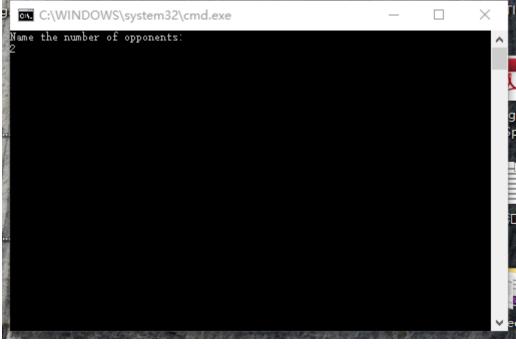
EEE102\_Assessment\_4\Task2\swordsman.cpp

- EEE102\_Assessment\_4\Task2\archer.h
- EEE102\_Assessment\_4\Task2\archer.cpp
- EEE102\_Assessment\_4\Task2\mage.h
- EEE102\_Assessment\_4\Task2\mage.cpp
- EEE102\_Assessment\_4\Task2\scorce.cpp

# 5. Testing

a) Ordinary Testing, using archer, 2 opponents randomly happen to be swordsman.



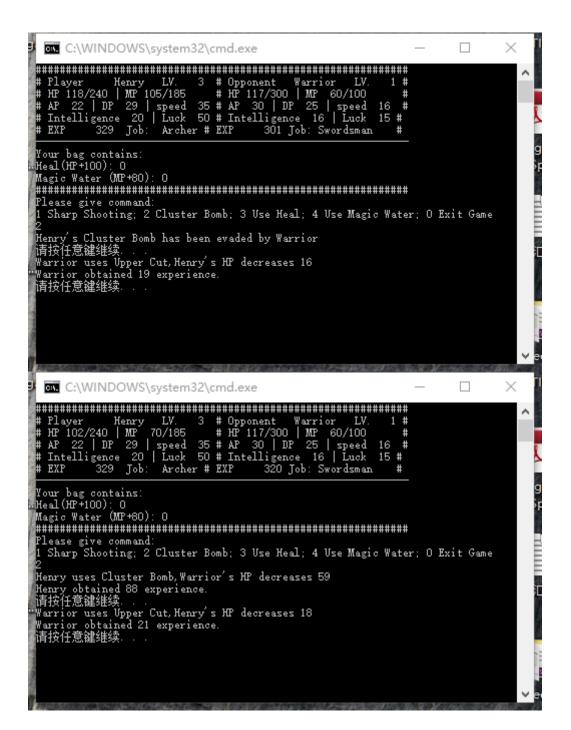


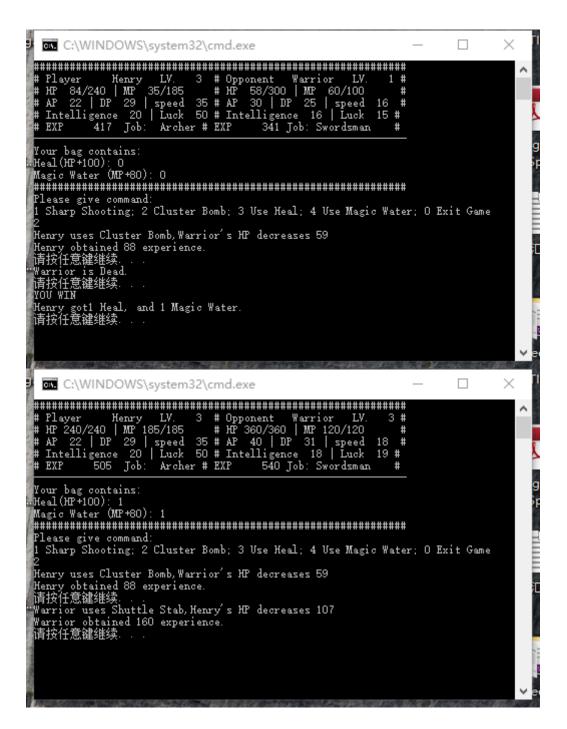


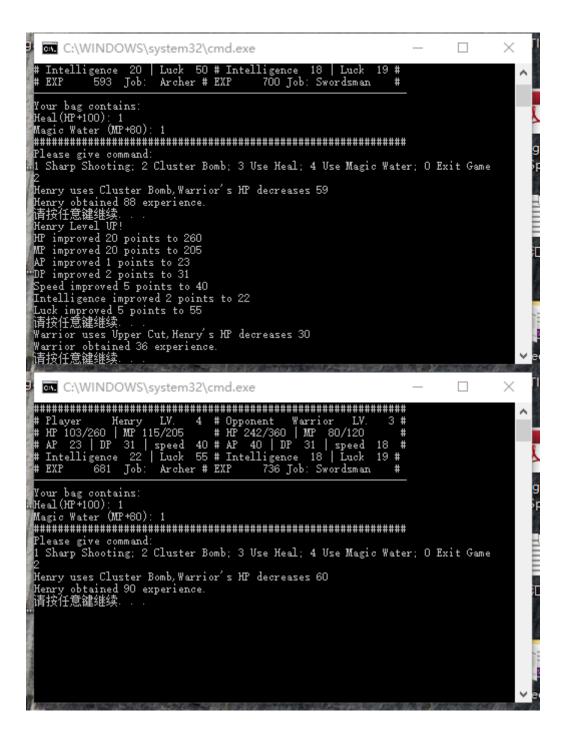


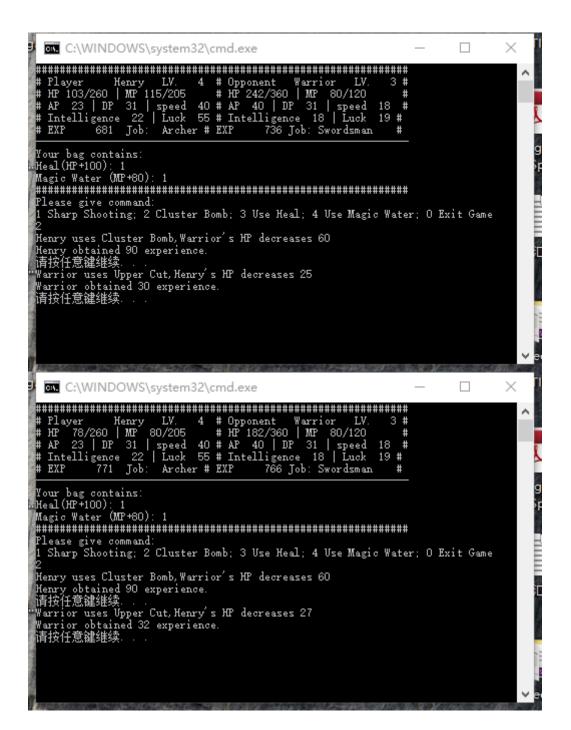




















b) Erroneous Input

