CS 162C Self Evaluation for Lab 7 – Player Array

Your name: Long To Lotto Tang	Date: 16/2/2022
Are you willing to allow your code to be used in example debugging demonstrations or documentation? ☒ Yes ☐ No	

Instructions – Part 1

This document is to be turned in alongside solution of this lab. You will use this document to indicate your status on the lab, as well as areas where you are struggling conceptually or in converting concept to code. Please use the space underneath each evaluation criteria to describe any errors you are receiving or challenges you are having implementing the required functionality for your code.

Functionality

Basic Expectations	***Completed***
Does the program compile and run?	Yes
Yes, the program compiles and runs without errors.	
Does the program run correctly and return the expected result?	Yes
Yes, the program runs correctly and returns the expected result.	
Are there comments explaining what the program and various functions are doing?	Yes
Yes, comments have been marked to explain the design intent of the code.	
Are all user inputs correctly validated?	Yes
Yes, checkClass() is used to validate the input string choice to return the string within the 4 classes. For the int type input, getInteger(min,max) is used to return the value within the range.	
Are class methods const where appropriate?	Yes
Yes, for class Person, getters (e.g. getFirstName, getLastName, getAge) have been declared with constant. For the class Player, the methods (getPlayerName, getPlayerAge, getClassName, getAction) have been declared with constant.	
Does your project include Person.hpp, Person.cpp, Player.hpp, Player.cpp, and main.cpp?	Yes
Yes, all 5 files have been included.	
Person Class	***Completed***
Does the Person class use initialization lists and inline functions in the header?	Yes
Yes, the Person class uses initialization lists and inline functions in Person.h.	
Does Person.cpp have a comment indicating it is empty deliberately?	Yes
Yes, Person.cpp includes a comment stating it is an empty file only.	
Player Class (parent)	***Completed***
Do both versions of the Player constructor work correctly?	Yes
Yes, both the default and overloaded constructor work correctly.	
Does the Player class dynamically create a new Person when it is instantiated?	Yes
Yes, a new Person is created dynamically by the following code: In Player.h (declare a new person in pointer) private: Person *person; In Player.cpp (default and overloaded constructor) Player:: Player(): person(new Person()){} Player:: Player(string firstName, string lastName, int age) : person(new Person(firstName, lastName, age)){}	

Does the destructor correctly delete the Person object inside of Player when Player is deleted?	Yes
Yes, I tried to use virtual ~Player() to delete the Person object inside of Player.	
Do the getPlayerName and getPlayerAge methods return the Person accessor results?	Yes

Yes, both functions return the Person accessor results (from getFirstName, getLastName and getAge)
`18 //Player's function getPlayerName (return the full name); getPlayerAge (get player's age)`
`19 string Player:: getPlayerName() const {return person->getFirstName() + " " + person->getLastName();}`
`20 int Player:: getPlayerAge() const {return person->getAge();}`	

Does the Player class include two pure virtual functions?	Yes
The Player class contains 3 pure virtual functions (e.g. ~Player(),getClassName() const and getAction() const).	

Character Classes (children)	***Completed***
Are all 4 child classes implemented?	Yes
Yes, 4 classes (Ranger,Wizard, Rogue, Priest) are all implemented.	
Are the child classes using public inheritance?	Yes
Yes, the classes are declared using public inheritance from parent Player.	
Do the child classes have constructors that properly initialize Player?	Yes
Yes, all child classes have overloaded constructor that properly initialize Player.	
Do the getClassName and getAction methods correctly override the parent method?	Yes
Yes, each child class will have its own getClassName() and getAction methods to override the parent method.	
Helper Functions	***Completed***
Is there a separate header and source file for the helper functions?	Yes
Yes, Week7B_HelperFunctions.h and Week7B_HelperFunctions.cpp are created.	
Did you include appropriate constants? Please list your constants below.	Yes

Yes, the constants are:
`7 int const MIN = 5;`
`8 int const MAX_PLAYER = 10;`
`9 int const MAX_AGE = 90;`	

Does displayPlayers safely accept a pointer to an array of Player objects and correctly output details for all child classes?	Yes

Yes, the parameters are passed with constant.
`17 void displayPlayers (Player * const *const gamerArray, int const num_player);`
 	
For the output of the information, the following code is implemented:
`for (int i = 0; i < num_player; i++)`
`{`
`//getClassName(), getAction() will override the virtual function from class Player respectively`
`cout << gamerArray[i]->getPlayerName() << " aged " << gamerArray[i]->getPlayerAge() << " playing a " << gamerArray[i]->getClassName() << " " << gamerArray[i]->getAction() << endl;`
`}`	

main	***Completed***
Does main contain a loop that creates the appropriate number of players?	No
No, the players are created dynamically in createPlayer().	
Does main only call displayPlayers once?	Yes
Yes, main() only calls displayPlayers once.	

Instructions – Part 2

Please answer the following questions, in your own words, regarding your experiences throughout this lab.

Experiential Review

What aspects of this lab did you find most challenging?

The declaration of the classes is the most challenging. At first, I mixed up the class composition and inheritance as I declared Player is the child of the class Person instead of following the instructions that require to be the instance of the Person through composition. The use of initialization list and assignment statement is quite difficult to understand its syntax currently, and I always forget to include "{}" for the methods and include extra ";" at the end of the methods. Furthermore, I am not sure about whether should I use virtual destructor with destructors for each derived classes to delete the objects. I used the above code in my program without any errors and warnings, but I am not sure the code is correct or not (do I need to delete individual derived class objects or simply the Player). Therefore, the declaration of the classes is the most challenging.

What concept/s from this lab do you feel you have the best grasp on now?

The techniques used in object-oriented programming. At first, I am quite confused in between composition and inheritance as I mentioned above. This evaluation form provides me an opportunity to do more research between the two and I have a better understanding about them. Furthermore, I am more familiar to use the dot or arrow notation to access the member functions or variables depending whether they are purely an object or a pointer.

This lab provides me the opportunity to declare classes through various of techniques. Frankly, I am still not familiar with the use of initialization list and assignment statements in the class declaration. The use of "::" and ":" is something quite confusing as the former will treat as the scope operator while the latter will be used in assignment of value (e.g. constructor) and for declaring in public or private for inheritance (class A : public B).

The other concepts like data validation, looping, logical operator are the best grasp on now.

Describe the concept of class composition and how it is used.

The concept of class composition is the process of building the complex large objects from the simpler ones. It is more common to describe class composition "has-a" relationship between 2 objects. As class can have any data type, including other classes, thus class composition can also be treated as creating class using other classes.

To declare class composition, the following code is used:

```
class A
{
   //body of class (e.g. constructor / methods)
}

class B
{
   A object1;

   public:
      B() : A();
};
```

The above code shows that is the complex object as it embeds class A to build up the new class. B can access the variables and methods from class A, with its own variables and methods, which are not accessible by the simpler class A.

What are the different parts of class inheritance and how do they work together?

Class inheritance is used when we know there is an "is a" relationship between a child and its parent class. For example, A student is a person, and Student as the child is a specialized class inherited from the parent class Person. Student can inherit the public, protected variables and methods from the parent while the child class can create its own variables and methods for its own purpose.

For instance, the code for the above example is as below:

```
class Person
{
   public:
     string name;
     //with other variables and methods

     Person(string name…)
};

class Student : public Person
{
   private:
     string home_address;
     string tel_num;

   public:
     Student (string name…., string home_address, string tel_num) : Person(name), home_address(home_address),
                tel_num(tel_num){}
     double GPA;

     //other Student related variables and methods
     void setGPA(GPA){this->GPA = GPA;}
     void getGPA(){return this->GPA};

};
```

As every student is a person, Student can inherit all the public and protected properties from Person (e.g. name, height, sex), while Student has its own specialized variables (e.g. private for personal information as it may contains private particulars, and public for general information like grade studying, GPA, student ID number etc.) and methods (e.g. calculate_average()). Therefore, the creation of a Student object (e.g. Peter as a student) can be done by calling the constructor and fill in the basic information (name, variables from Person) and its own variables (home_address) to implement both the methods from the parent and child class.

What is the purpose of an abstract class?

Abstract class is a class that is designed to be specifically used as the base class with at least 1 pure virtual function in the class declaration. The derived classes must provide a definition for the pure virtual function, otherwise the derived classes would become an abstract class as well. We cannot use abstract class as a parameter type or function return type, as well as declaring an object of an abstract class, but we can declare pointers and references to an abstract class.

Using abstract class can make the code reusable and extendable by abstraction. As the derived classes may have the same methods but different output value, the abstract class provides the basis or prototype of the derived classes such that the derived classes can inherit the basic variables while the use of virtual function allows the derived classes to deliver different output.

Describe the differences and similarities between inheritance and composition and what kind of things each might be used for.

Inheritance and composition are 2 programming techniques to establish relationship between classes and objects.

For composition, it will create an object within another object while inheritance is the functionality that one object acquires the characteristics of one or more other objects. Object composition is defined dynamically at run-time while class inheritance is defined at compile-time. The internal details for composition are not exposed to each other since 2 objects interact through the public interfaces, while the parent from inheritance will expose both public and protected members. Therefore, access control can be done by composition but not in inheritance (cannot change the base class for composition), which also implies that data encapsulation can be done in composition but difficult in inheritance (possible for setting in private).

In short, when we consider the 2 concepts in plain English, for instance, “a student is a person” but not “a person has a student”, we can simplify the consideration in choosing between inheritance and composition.

Describe the concept of polymorphism and how it is used.

Polymorphism is that when there is hierarchy of classes by inheritance, the result for calling a method behaves differently depending on the type of the object invoking the call. In this lab, function overriding with virtual function is used as the base class Player declared the virtual function for getClassName() and getAction(), the 4 derived classes declared the same methods but with different return values. Such that when calling the function, the compiler determines which function to be invoked at runtime.

For the compile time polymorphism, function overloading is one of the forms. By having the same name of the functions with different parameters, these functions will be treated as differently and they are said to be overloaded. The return of the overloaded functions will depend on which functions to be called (based on the parameters).