Part B - Foundations

Derived Classes

Workshop 7

In this workshop, you are to inherit one class from another class and support a class with a global function.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities:

- to define a derived class with reference to a base class
- to shadow a member function of a base class with a member function of a derived class
- to access a shadowed member function in a base class
- to define a helper function in terms of a member function of the supported class
- to implement a friendship between a helper function and the class it supports
- to describe to your instructor what you have learned in completing this workshop

Submission Policy

The *"in-lab"* section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the *"in-lab"* portion of the workshop during that period, ask your instructor for permission to complete the *"in-lab"* portion after the period. If you do not attend the workshop, you can submit the *"in-lab"* section along with your *"at-home"* section (with a penalty; see below). The *"at-home"* portion of the lab is **due on the day of your next scheduled workshop** (at 23:59).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly back up your work.

Late submission penalties:

- *“in-lab”* submitted late, with *“at-home”* submitted on time: maximum of 20/50 for *“in-lab”* and maximum of 50/50 for “*at-home*”*;*

- *“in-lab”* on time, with *“at-home”* and *“reflection”* submitted at most one week late: maximum of 50/50 for *“in-lab”* and maximum of 20/50 for *“at-home”*;

- workshop late for at most one week: *“in-lab”*, *“at-home”* **and** *“reflection”* must all be submitted for maximum of 50/100;

- workshop late for more than one week: *“in-lab”*, *“at-home”* **and** *“reflection”* must all be submitted for maximum of 30/100;

- if any of *“in-lab”*, *“at-home”* or *“reflection”* is missing, the mark will be **zero**.

IN-LAB: The Hero Class (50%)

Design and code a class named **Hero** that holds information about a fictional hero character. Place your class definition in a header file named **Hero.h** and your function definitions in an implementation file named **Hero.cpp**. Include in your design all of the statements necessary to compile and to run your code successfully under a standard C++ compiler.

Heros have a simple model to determine who wins in a fight. Each hero has an integer health number. If the health is 0, the Hero is not alive. The health can go up to a maximum as stored in the `m_maximumHealth` variable**.**

During a fight a Hero can attack another Hero. Each Hero has their own specific attack strength. The attack inflicts damage on the receiver. In a fight where Hero 1 attacks Hero 2, and Hero 2 takes damage.

Damage (*for hero 2****) = - Attack (****for hero 1****)***

Damage is subtracted from the health. The winner is whoever is alive when the opponent is dead.

The Hero class must contain the following data

Member variable name	Data type	Default value	Description
`m_name[41]`	`char`	`""`	`String to hold the hero's name. Maximum length 40 chars + '\0' terminator.`
`m_health`	`int`	`0`	`The hero's health, from 0 to the value specified in m_maximumHealth`
`m_maximumHealth`	`unsigned`	`0`	`The maximum health points of the Hero.`
`m_attack`	`unsigned`	`0`	`The attack strength of the hero.`

Upon instantiation, a Hero object may receive **no** information, or may receive **three** values:

- A string which can be nullptr or empty, giving the name
- The Maximum Health value
- The attack value

The constructor should initialize the Health to maximumHealth.

Your design must also include the following member functions and one helper operator:

Function Declaration	**Description**
bool isEmpty() const	Returns **true** if the object is in the safe empty state; **false** otherwise.
unsigned getAttack() const	Returns the attack strength for the hero. The return value can be 0 for an object in the safe empty state. *Note: this means you do not have to explicitly check for it!*
void display (std::ostream&) const	prints the hero's name to the output stream. If the current object is empty, this function should print nothing.
bool isAlive() const	Returns true if the Hero is alive, false if not. A hero is alive if his/her health is > 0.
void respawn()	Restore the Hero's health to his/her maximum value.

Global helper functions

Implement the following helper function in the file fight.cpp.

`friend void`	A function to inflict 1 unit of damage on

`apply_damage(Hero& A, Hero& B)`	each Hero.
	It should deduct from the Hero's health, according to the damage formula given previously. For example, if Hero A and B attack each other,
	
	`A.m_health -= B.m_attack;`
	`B.m_health -= A.m_attack;`
	
	Also put appropriate checks for 0 values in the function. Note that the Hero's health should not decrease to below 0.

A function in fight.cpp has been given to handle a complete fight between two heroes, A and B. It is implemented as the "*" operator. Note the following features of the function.

- The inputs are **const Hero&**. This ensures that the heroes are unharmed during the tournament. The first thing the function does is make a copy of them to local variables. Note that you do not need a copy constructor...because the class does not contain any dynamically allocated resources.

- It returns a reference to the winner. This allows you to pass the return value of the function to another function call.

The following program uses your Hero class and produces the output shown below:

```
#include <iostream>
#include "Hero.h"
#include "fight.h"

using namespace std;


 int main ()
 {

   cout << endl << "Greek Heros";
   Hero hercules         ("Hercules",  32, 4);
   Hero theseus          ("Theseus",   14, 5);
   Hero oddyseus         ("Odysseus",  15, 3);
   Hero ajax             ("Ajax",      17, 5);
   Hero achilles         ("Achilles",  20, 6);
   Hero hector           ("Hector",    30, 5);
   Hero atalanta         ("Atalanta",  10, 3);
   Hero hippolyta        ("Hippolyta", 10, 2);



   cout << endl << "Quarter Finals" << endl;
   const Hero& greek_winner1 = achilles * hector;
   const Hero& greek_winner2 = hercules * theseus;
   const Hero& greek_winner3 = oddyseus * ajax;
   const Hero& greek_winner4 = atalanta   * hippolyta;

   cout << endl << "Semi Finals" << endl;
   const Hero& greek_winner_semifinal1 = greek_winner1  * greek_winner2;
   const Hero& greek_winner_semifinal2 = greek_winner3  * greek_winner4;

   cout << endl << "Finals" << endl;
   greek_winner_semifinal1 * greek_winner_semifinal2;
```

```
Greek Heros
Quarter Finals
Battle! Achilles vs Hector : Winner is Hector in 4 rounds.
Battle! Hercules vs Theseus : Winner is Hercules in 4 rounds.
Battle! Odysseus vs Ajax : Winner is Ajax in 3 rounds.
Battle! Atalanta vs Hippolyta : Winner is Atalanta in 4 rounds.

Semi Finals
Battle! Hector vs Hercules : Winner is Hector in 7 rounds.
```

```
Battle! Ajax vs Atalanta : Winner is Ajax in 2 rounds.

Finals
Battle! Hector vs Ajax : Winner is Hector in 4 rounds.
```

In Lab SUBMISSION

If not on matrix already, upload fight.cpp, fight.h, Hero.h, Hero.cpp, w7_in_lab.cpp to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

~profname.proflastname/submit 244_w7_lab <ENTER>

At Home: SuperHero Class (40%)

Derive a class named SuperHero from the Hero class that you designed in the *in_lab* section. Include in your design all of the statements and keywords necessary to compile and to run your code successfully under a standard C++ compiler.

SuperHeros behave the same as Heros, except that they have SuperPowers! A SuperHero still has a regular normal attack, and health...but he/she can also use his SuperPower to attack, and to defend—but only against other superheros. ***When a SuperHero fights against a Hero, he/she does not use his SuperPowers. They only uses their regular Hero attack.***

Upon instantiation, a SuperHero object may receive no information, or it may receive **five** values:

- The **name**, **maximum health** and **attack** values similar to the Hero class.
- An unsigned integer for the **SuperPowerAttack** strength of the SuperHero.
- A unsigned integer representing a The **SuperPowerDefend** strength for the SuperHero.

Fighting for the SuperHero is similar to the Hero class. Damage is calculated differently though. When Hero1 attacks Hero2

Damage (*Hero 2)* = **SuperPowerAttack** (*Hero 1)* – **SuperPowerDefend** (*Hero 2)*.

Build constructors to handle these instantiation types. Invalid input sets the object in the safe, empty state.

SuperHero should add the following new *member variables*.

`m_superPowerAttack`	`unsigned`	`0`	`The SuperPower attacking strength`
`m_superPowerDefend`	`unsigned`	`0`	`The SuperPower defense strength`

SuperHero also adds the following *member functions*: Note that they have the ***same names*** as the functions in Hero. They are supposed to shadow the functions in the Hero class.

- unsigned getAttack() const -- This method should return the SuperPower Attack of the superhero instead of the regular Attack value.
- Note that this method has the same name and prototype as the one in Hero.

You will also need a new global helper function, **apply_damage.** It will compute the damager both SuperHeros inflicts on each other. This should have the following prototype

```
friend void apply_damage ( SuperHero& A,  SuperHero& B);
```

The following code uses your Hero and SuperHero classes and produces the output shown under it:

```
int main ()
{

  cout << endl;
  line(60); // print some dashes

  cout << endl << "Greek Heros";
  Hero hercules         ("Hercules",  32, 4);
  Hero theseus          ("Theseus",   14, 5);
  Hero oddyseus         ("Odysseus",  15, 3);
  Hero ajax             ("Ajax",      17, 5);
  Hero achilles         ("Achilles",  20, 6);
  Hero hector           ("Hector",    30, 5);
  Hero atalanta         ("Atalanta",  10, 3);
  Hero hippolyta        ("Hippolyta", 10, 2);

  cout << endl << "Quarter Finals" << endl;
  const Hero& greek_winner1 = achilles * hector;
  const Hero& greek_winner2 = hercules * theseus;
  const Hero& greek_winner3 = oddyseus * ajax;
  const Hero& greek_winner4 = atalanta   * hippolyta;

  cout << endl << "Semi Finals" << endl;
  const Hero& greek_winner_semifinal1 = greek_winner1  * greek_winner2;
  const Hero& greek_winner_semifinal2 = greek_winner3  * greek_winner4;

  cout << endl << "Finals" << endl;
  const Hero& greek_final = greek_winner_semifinal1 * greek_winner_semifinal2;
```

```
  line(60);
  cout << endl << "Comic book SuperHeros";

  SuperHero superman    ("Superman",  50, 9, 10, 9) ;
  SuperHero hulk        ("The_Hulk",   70, 6, 20, 3) ;
  SuperHero wonderwoman ("WonderWoman",   80, 5, 15, 10) ;
  SuperHero raven       ("Raven",   30, 10, 12, 5) ;

  cout << endl << "Semi Finals" << endl;
  const SuperHero& comic_winner1 = superman * hulk;
  const SuperHero& comic_winner2 = wonderwoman * raven;
  cout << endl << "Finals" << endl;
  const SuperHero& comic_final = comic_winner1 * comic_winner2;

  line(60);
  cout << endl << "Best Greeks Hero vs Best Comic Book SuperHero" << endl;
  greek_final * comic_final;
```

```
------------------------------------------------------------
Greek Heros
Quarter Finals
Battle! Achilles vs Hector : Winner is Hector in 4 rounds.
Battle! Hercules vs Theseus : Winner is Hercules in 4 rounds.
Battle! Odysseus vs Ajax : Winner is Ajax in 3 rounds.
Battle! Atalanta vs Hippolyta : Winner is Atalanta in 4 rounds.

Semi Finals
Battle! Hector vs Hercules : Winner is Hector in 7 rounds.
Battle! Ajax vs Atalanta : Winner is Ajax in 2 rounds.

Finals
Battle! Hector vs Ajax : Winner is Hector in 4 rounds.
------------------------------------------------------------
Comic book SuperHeros
Semi Finals
Fight! Superman vs The_Hulk : Winner is The_Hulk in 5 rounds.
Fight! WonderWoman vs Raven : Winner is WonderWoman in 3 rounds.

Finals
Fight! The_Hulk vs WonderWoman : Winner is WonderWoman in 6 rounds.
------------------------------------------------------------
Best Greeks Hero vs Best Comic Book SuperHero
Battle! Hector vs WonderWoman : Winner is WonderWoman in 6 rounds.
```

At-Home Reflection (10%)

Answer the following questions and place them in a file called reflect.txt.

1. Why did we choose m_health to be a signed integer? (Hint: what happens if the Hero has a health of 1, and someone does 2 points of damage to them.)

2. Does the Hero class need to know about the existence of the class SuperHero? (Hint: do a search in Hero.cpp, does the word "SuperHero" appear anywhere in it?) How about the reverse, does SuperHero know about the Hero class?

3. The program prints out "AncientBattle!" when 2 Heros fight. It prints out "SuperFight!" when 2 SuperHeros fight. When you try to make a Hero fight a SuperHero, what did it print out?

At-Home Submission:

To submit the *at-home* section, demonstrate execution of your program with the exact output as in the example above. Upload the following files to your matrix account: **Hero.h, Hero.cpp, SuperHero.h, SuperHero.cpp, fight.h, fight.cpp, w7_athome.cpp**. Compile and run your code and make sure everything works properly. To submit, run the following script from your account (and follow the instructions):

~profname.proflastname/submit 244_w7_home<ENTER>

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.