DERIVED CLASSES

Workshop 7 (out of 10 marks – 3.75% of your final grade)

In this workshop, you are to define a base class, inherit a derived class from that base class and define helper functions to support the two classes.

LEARNING OUTCOMES

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

- to inherit a derived class from 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 that accesses a member function of the supported class
- to implement 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 that is two days before your next scheduled workshop (23:59:59).

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

You are responsible to back up your work regularly.

LATE SUBMISSION PENALTIES:

- *In-lab* portion submitted late, with *at-home* portion: **0** for *in-lab*. Maximum of 7/10 for the entire workshop.
- If any of *in-lab*, *at-home* or *reflection* portions is missing, the mark for the workshop will be **0**/10.

IN-LAB: THE BASE CLASS (30%)

Design a base class named `Hero`, in namespace `sict`. This class 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.

`Hero`es battle under a simple model. Each `Hero` has a non-negative integer health number. If the health is 0, the `Hero` is not alive. During a battle `Hero`es attack one another. Each `Hero` has their own specific attack strength. Each attack inflicts damage on the attacked `Hero`. The inflicted damage is subtracted from the attacked `Hero`'s health. The winner of the battle is the `Hero` who remains alive when the other `Hero` is dead. A `Hero` is alive as long as their health number is positive-valued. A battle ends in a draw after `max_rounds` rounds.

A `Hero` object contains the following data:

- The name of the `Hero` – up to 40 characters (excluding the null byte)
- The health of the `Hero` – a positive-valued integer
- The attack strength of the `Hero` – a positive-valued integer

Upon instantiation, a `Hero` object receives no information or information on all three values. If the information received is valid, the object accepts them. Otherwise, the object assumes a safe empty state.

Your design includes the following member functions:

`void operator-=(int attack)`: an overloaded operator that receives an attack strength and, if that strength is positive-valued, deducts that

strength from the current object's health. If the attack strength received is not positive-valued, this operator does nothing. If the deduction drops the current object's health below 0, this operator resets its health to 0.

`bool isAlive() const`: a query that returns `true` if the current object is healthy and `false` otherwise.

`int attackStrength() const`: a query that returns the attack strength of the current object. If the object is in a safe empty state, this function returns 0.

Your design also includes two helper operators, which support your class:

`ostream& operator<<(ostream& os, const Hero& hero)`: a friend that inserts the name of `hero` into stream `os` and returns a reference to that stream. If `hero` is in a safe empty state, this function displays the message:

```
No hero
```

`const Hero& operator*(const Hero& first, const Hero& second)`: a non-friend that returns an unmodifiable reference to the winner of the battle between the `Hero`es after `max_rounds` rounds. This function displays the names of the battle participants as shown in the sample output below, makes local copies of the participants, determines the damage that each inflicts on the other in a single attack and battles until either one of the participants dies or the maximum number of rounds is reached. In each round, this function deducts the damage inflicted on a `Hero` by the other `Hero`. Finally, this function displays the name of the winner. In the case of a draw, this function assumes arbitrarily that the left operand (`first`) has won. This function returns a reference to the winner object.

Using the sample implementation of the `w7_in_lab.cpp` main module listed below, test your code and make sure that it works. The expected output from your program is listed below this source code. The output of your program should match **exactly** the expected one.

In-Lab Main Module

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

using namespace std;
using namespace sict;

int main()
{

    cout << "Greek Heroes";
    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;

    return 0;
}
```

In-Lab Output

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

Semi Finals
```

```
Ancient Battle! Hector vs Hercules : Winner is Hector in 7 rounds.
Ancient Battle! Ajax vs Atalanta : Winner is Ajax in 2 rounds.

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

IN-LAB SUBMISSION

If not on matrix already, upload `Hero.h`, `Hero.cpp` and `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 (use your professor's Seneca userid to replace `profname.proflastname`):

~profname.proflastname/submit 244_w7_lab<ENTER>

and follow the instructions.

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

AT-HOME: SUPERHERO CLASS (30%)

Derive a class named SuperHero from the Hero class that you coded for 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.

Super Heroes behave the same as Heroes, except that Super Heroes have Super Powers! A SuperHero has a regular normal attack strength, and health, but can also use their super power to attack with a bonus, and to defend themselves —but only against another SuperHero. ***When a SuperHero fights against a Hero, the SuperHero does not use its super power, but only uses its regular Hero attack strength.***

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

- The address of a C-style string containing the name of the SuperHero,
- The **health** of the SuperHero,
- The **attack** strength of the SuperHero in attacks on a Hero,
- The super power attack **bonus** of the SuperHero,
- The **defend** strength of the SuperHero.

The fight model for a SuperHero is the same as the fight model for a Hero, but the damage in any single round is calculated differently. When a SuperHero attacks another SuperHero, the attack strength of the attacking SuperHero is its attack strength plus its bonus attack strength. Moreover, when a SuperHero is attacked by another SuperHero, the damage to the attacked SuperHero is the attack strength of the attacking SuperHero minus the defend strength of the attacked SuperHero.

```
Damage(A) = (Attack of B + AttackBonus of B) – DefendStrength of A
Damage(B) = (Attack of A + AttackBonus of A) – DefendStrength of B
```

Your SuperHero class includes the two constructors described above and the following member functions:

int attackStrength() const: a query that returns the attack strength of the current object including its super power bonus. If the object is in a safe empty state, this function returns 0.

int defend() const: a query that returns the defend strength of the current object. If the object is in a safe empty state, this function returns 0.

Your design also includes a helper operator, which supports your class:

const SuperHero& operator*(const SuperHero& first, const SuperHero& second): a non-friend that returns an unmodifiable reference to the winner of the battle between the SuperHeroes after max_rounds rounds. This function displays the names of the battle participants as shown in the sample output below, makes local copies of the participants, determines the damage that each inflicts on the other in a single attack and battles until either one of the participants dies or the max-

imum number of rounds is reached. In each round, this function de-
ducts the damage inflicted on a SuperHero by the other SuperHero.
Finally, this function displays the name of the winner. In the case of a
draw, this function assumes arbitrarily that the left operand (first)
has won. This function returns a reference to the winner object.

The following program (w7_at_home.cpp) uses your SuperHero and Hero classes and
produces the output listed below.

```cpp
#include <iostream>
#include "Hero.h"
#include "SuperHero.h"

using namespace std;
using namespace sict;

void line(int width) {
        cout.width(width - 1);
        cout.fill('-');
        cout << '-';
        cout.fill(' ');
}

int main() {
  line(60);

  cout << endl << "Greek Heroes";
  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, 1, 9) ;
```

```cpp
    SuperHero hulk          ("The_Hulk",      70, 6, 14, 3) ;
    SuperHero wonderwoman ("WonderWoman",    80, 5, 10, 10) ;
    SuperHero raven         ("Raven",         30, 10, 2, 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 Heroes
Quarter Finals
Ancient Battle! Achilles vs Hector : Winner is Hector in 4 rounds.
Ancient Battle! Hercules vs Theseus : Winner is Hercules in 4 rounds.
Ancient Battle! Odysseus vs Ajax : Winner is Ajax in 3 rounds.
Ancient Battle! Atalanta vs Hippolyta : Winner is Atalanta in 4 rounds.

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

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

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

Ancient Battle! Hector vs WonderWoman : Winner is WonderWoman in 6 rounds.
```

REFLECTION (40%)

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

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

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

3. The program prints out "Ancient Battle!" when 2 Heroes fight. It prints out "Super Fight!" when 2 SuperHeroes fight. When you tried to make a Hero fight a SuperHero, what did it print out?

4. True or False: are the following a valid program definition for main?

```
int main()
{
   Hero("Achilles", 20, 6) * Hero("Hector", 30, 5);
}
```

how about this?

```
int main()
{
  (Hero("Achilles", 20, 6) * Hero("Hector", 30, 5))
                         *
  (Hero("Atalanta", 10, 3) * Hero("Hippolyta", 10, 2));
}
```

QUIZ REFLECTION:

Add a section to reflect.txt called Quiz X Reflection. Replace the X with the number of the last quiz that you received and list the numbers of all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

At-Home Submission:

If not already on matrix, upload the following files to your matrix account: **Hero.h, Hero.cpp, SuperHero.h, SuperHero.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>

and follow the instructions.

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