Homework 2

Focus

- **Functions**
- **Structs**
- **Vectors**
- **I/O**

Problem:

We will model a game of medieval times. Our world is filled with warriors. Naturally what warriors like to do is fight. To the death. So we happily let them.

Each warrior starts out with a name and a certain amount of strength. Each time he fights, he loses some strength. (He gets to keep his name.) If his opponent is stronger than he is, then he loses *all* of his strength, in which case he is dead, or at the very least pretty useless as a fighter. Otherwise he loses as much strength as his opponent had. Of course, if he and his opponent had the same strength then they are both losers.

Even losers are allowed to pick a fight. It doesn't require having any strength in order to do battle with someone else. Not that you stand much of a chance of winning anything, but perhaps it's worth getting beaten (again) just to have those 15 seconds of fame.

Your program will read in a file of commands. There are three types of commands:

- `Warrior` creates a new warrior with the specified name and strength.
- `Battle` causes a battle to occur between two warriors.
- `Status` lists all warriors, alive or dead, and their strengths.

A sample input file looks like:

```
Warrior Jim 42

Warrior Lancelot 15

Warrior Arthur 15

Warrior Torvalds 20

Warrior Gates 8

Status

Battle Arthur Lancelot
```

```
Battle Jim Lancelot

Battle Torvalds Gates

Battle Gates Lancelot

Status
```

The name of the input file will be "warriors.txt". Note that the commands do not have to appear in that order. The only requirement is that a Battle command cannot appear until the specified warriors have been seen.

The Status command displays how many warriors there, then displays each one with his strength. The Warrior command does not display anything. The Battle command displays one line to says who is fighting whom and a second line to report the results, as shown below.

The output (which would display on the screen) for this file should look like:

```
There are: 5 warriors

Warrior: Jim, strength: 42

Warrior: Lancelot, strength: 15

Warrior: Arthur, strength: 15

Warrior: Torvalds, strength: 20

Warrior: Gates, strength: 8

Arthur battles Lancelot

Mutual Annihilation: Arthur and Lancelot die at each other's hands

Jim battles Lancelot

He's dead, Jim

Torvalds battles Gates

Torvalds defeats Gates

Gates battles Lancelot

Oh, NO! They're both dead! Yuck!
```

```
There are: 5 warriors

Warrior: Jim, strength: 42

Warrior: Lancelot, strength: 0

Warrior: Arthur, strength: 0

Warrior: Torvalds, strength: 12

Warrior: Gates, strength: 0
```

Homework 3

Focus:

- Object-oriented programming
    - Data Hiding
    - Encapsulation
    - Delegation
    - Output operator Overloading

Problem:

We will expand our Warrior a little. Each Warrior will have a weapon. He is "born" with it, i.e. the weapon is created together with the warrior. It can only be accessed by him. It provides him with his strength. In battle, weapons lose their edge and weaken. When a Warrior's weapon loses all of its strength, the Warrior himself dies.

Implementation

- Remember that we are using *data hiding*. Therefore, every field, aka member variable, *must* be private.
- What are the *types* of things in the problem? We will need a class for each type.
- What do the things / types *do*? These "behaviors" should be represented as *methods*.
- Weapons have both a name and a strength. The weapon is created together with the Warrior and cannot be accessed by anyone else.
- Note that the input file changed a little, compared to the previous assignment. When a Warrior is created, instead of simply specifying his name and strength, the Warrior command specifies the Warrior's name as well as his Weapon's name and *its* strength.
- The Status report will is also modified to show the name of the Warrior's Weapon.
- ***No one*** can access a warrior's weapon **except the warrior himself**. But the weapon is what actually holds the warrior's strength. How does this effect the programming? Any time the

code needs to know or change the warrior's strength, **the warrior then "asks" the weapon** what the strength is or tells the weapon that the strength needs to be changed. This represents the idea of ***delegation***. We will see this concept frequently, where one object requests that another object do some task.

- It is in fact unnecessary for any code other than a Warrior to even know about the Weapon. We will enforce this by *nesting* the definition of the Weapon class inside the Warrior class. To make sure that no code other than Warrior's makes use of Weapon, we need to make the class private.
- The name of the input file will be "warriors.txt".
- One last implementation detail. to display the information about an object, whether it is a warrior or a weapon, we will use that object's output operator.

Input

Our sample input file might now look like:

```
Warrior Jim Glamdring 42

Warrior Lancelot Naegling 15

Warrior Arthur Excalibur 15

Warrior Torvalds Narsil 20

Warrior Gates Orcrist 8

Status

Battle Arthur Lancelot

Battle Jim Lancelot

Battle Torvalds Gates

Battle Gates Lancelot

Status
```

Output

The corresponding output would be:

```
There are: 5 warriors

Warrior: Jim, weapon: Glamdring, 42
```

```
Warrior: Lancelot, weapon: Naegling, 15
Warrior: Arthur, weapon: Excalibur, 15
Warrior: Torvalds, weapon: Narsil, 20
Warrior: Gates, weapon: Orcrist, 8
Arthur battles Lancelot
Mutual Annihilation: Arthur and Lancelot die at each other's hands
Jim battles Lancelot
He's dead, Jim
Torvalds battles Gates
Torvalds defeats Gates
Gates battles Lancelot
Oh, NO! They're both dead! Yuck!
There are: 5 warriors
Warrior: Jim, weapon: Glamdring, 42
Warrior: Lancelot, weapon: Naegling, 0
Warrior: Arthur, weapon: Excalibur, 0
Warrior: Torvalds, weapon: Narsil, 12
Warrior: Gates, weapon: Orcrist, 0
```

Homework 4

Focus

- Classes
- Association
- As always: good coding

Problem:

We will [continue to] model a game of medieval times. Our world is filled with not only warriors but also nobles. Nobles don't have much to do except do battle with each other. (We'll leave the feasting and other entertainments for add-ons.) Warriors don't have much to do except hire out to a noble and fight in his behalf. Of course the nobles are pretty wimpy themselves and will lose if they don't have warriors to defend them. How does all this work?

- Warriors start out with a specified strength.
- A battle between nobles is won by the noble who commands the stronger army.

- The army's strength is simply the combined strengths of all its warriors.
- A battle is to the death. The losing noble dies as does his warriors.
- The winner does not usually walk away unscarred. All his men lose a portion of their strength equal to the ratio of the enemy army's combined strenth to their army's. If the losing army had a combined strength that was 1/4 the size of the winning army's, then each soldier in the winning army will have their own strength reduced by 1/4.

Hiring and Firing

- Warriors are hired and fired by Nobles. Lack of adequate labor laws, have left the Warriors without the ability to quit, nor even to have a say on whether or not a Noble can hire them.
- However it is possible that an attempt to hire or fire may fail. Naturally the methods should not "fail silently". Instead, they will return true or false depending on whether they succeed or not.
- A Warrior can only be employed by one Noble at a time and *cannot* be hired away if he is already employed.
- As noted below, Nobles who are dead can neither hire nor fire anyone. (Note this will *implicitly* prevent dead Warriors from being hired.)
- When a warrior is fired, he is no longer part of the army of the Noble that hired him. He is then free to be hired by another Noble.
    - How do you remove something from a vector.
    - While there are techniques that make use of iterators, we have not yet discussed iterators so you will not use them here. (As a heads up, if you see a technique that requires you to call a vector's begin() method, that is using iterators. Don't use it.)
    - While it may seem a slight burden, certainly it does not require more than a simple loop to remove an item from a vector. No <u>do not</u> do something silly like create a whole new vector.
    - Soon we will cover iterators and then you will be freed from these constraints. Patience, please.

Death

It's a sad topic, but one we do have to address.

- People die when they lose a battle, whether they are a Nobles or Warriors.
- Nobles who are dead are in no position to hire or fire anyone. Any attempt by a dead Lord to hire someone will simply fail and the Warrior will remain unhired.
- However curiously, as has been seen before, Nobles can declare battle even though they are dead.
- Note that when a Noble is created he does not have any strength. At the same time he is obviously alive. So lack of strength and being dead are clearly *not* equivalent.

A test program and output are attached. Note that the output shown is what you are <u>expected</u> to generate. Pardon us, we don't like limiting your creativity, but having your output consistent with ours makes the first step of grading a bit easier. And also helps you to be more confident that your code works.

Programming Constraints

What would a homework assignment be without unnecessary and unreasonable constraints?

Your classes must satisfy the following:

- The **battle method** will announce who is battling whom, and the result (as shown in the example output).
    - If one or both of the nobles is already dead, just report that.
      The "winner" doesn't win anything for kicking around a dead guy. And his warriors don't use up any strength.
    - Look at the output for the sample test program to see what you should be displaying.
- A noble's army is a **vector of pointers** to warriors. Warriors will be ordered in the army by the order in which they were *hired*. Be sure to maintain that order when a Warrior gets fired.

Homework 5

Focus

- Dynamic memory
- But this does *not* involve copy control. No destructor, copy constructor or assignment operator.

Update

- Do **NOT** make the Noble class a friend of the Warrior class.
- Do **NOT** put a pointer in the Warrior class to the Noble class.

Problem:

Building on previous assignments, we will be reading a file of commands to create Nobles and Warriors, and sending them off to battle.

Key differences:

- Each time a warrior or a noble is defined, we will create it on the heap.
- We will keep track of the nobles in a vector of pointers to nobles.
- We will keep track of *all warriors* using a vector of pointers to warriors.

The input file will be named "`nobleWarriors.txt`".

Commands

- Noble. Create a Noble on the heap.
- Warrior. Create a Warrior on the heap.
- Hire. Call the Noble's hire method.
- Fire. Call the Noble's fire method.

- Battle. Call the Noble's battle method.
- Status. The status command shows the nobles, together with their armies, as we did previously. In addition, it will show separately the warriors who do not currentle have a employer
- Clear. Clear out all the nobles and warriors that were created.

Our application is going to rely on each Noble having a unique name and each Warrior having a unique name. Otherwise, how would we be sure who we were hiring (or firing). Note that this is not a requirement of the Noble and Warrior classes themselves, just of this particular *use* of them, i.e. our application.

Whenever you are displaying a Noble or a Warrior, you will use the **output operator** for the class.

Handle errors!

Previously we promised that all of the commands we gave you in the input would be valid. Now we would like you to take some responsibility for checking the input. First, we still guarantee that the format of the file will be correct. That means that the Warrior command will always have a name and a strength. The Battle command will always have two names. The Status command will not have any other information on it than just the word Status.

However, you will need to detect and report any issues indicating inconsistencies, such as:

- Noble command: if a Noble with a given name already exists.
- Warrior command:if a Warrior with a given name already exists.
- Hire command: If a Noble tries to hire a Warrior and either of them do not exist.
- Fire command: If a Noble tries to fire a Warrior and either the Noble does not exist or does not have the Warrior by that name in his army.
- Battle command: If a Noble initiates a battle with another Noble, but one or the other does not exist.

We have not specified the format of these error messages, so we'll leave that up to you. (You get to be creative!)

Example input file (no errors):

```
Noble King_Arthur

Noble Lancelot_du_Lac

Noble Jim

Noble Linus_Torvalds

Noble Bill_Gates

Warrior Tarzan 10
```

```
Warrior Merlin 15

Warrior Conan 12

Warrior Spock 15

Warrior Xena 20

Warrior Hulk 8

Warrior Hercules 3

Hire Jim Spock

Hire Lancelot_du_Lac Conan

Hire King_Arthur Merlin

Hire Lancelot_du_Lac Hercules

Hire Linus_Torvalds Xena

Hire Bill_Gates Hulk

Hire King_Arthur Tarzan

Status

Fire King_Arthur Tarzan

Status

Battle King_Arthur Lancelot_du_Lac

Battle Jim Lancelot_du_Lac

Battle Linus_Torvalds Bill_Gates

Battle Bill_Gates Lancelot_du_Lac

Status
```

```
Clear
```

```
Status
```

Example output:

```
Status
======
Nobles:
King_Arthur has an army of 2
        Merlin: 15
        Tarzan: 10
Lancelot_du_Lac has an army of 2
        Conan: 12
        Hercules: 3
Jim has an army of 1
        Spock: 15
Linus_Torvalds has an army of 1
        Xena: 20
Bill_Gates has an army of 1
        Hulk: 8
Unemployed Warriors:
NONE
You don't work for me anymore Tarzan! -- King_Arthur.
Status
======
Nobles:
King_Arthur has an army of 1
        Merlin: 15
Lancelot_du_Lac has an army of 2
        Conan: 12
        Hercules: 3
Jim has an army of 1
        Spock: 15
Linus_Torvalds has an army of 1
        Xena: 20
Bill_Gates has an army of 1
        Hulk: 8
Unemployed Warriors:
Tarzan: 10
King_Arthur battles Lancelot_du_Lac
Mutual Annihalation: King_Arthur and Lancelot_du_Lac die at each other's hands
Jim battles Lancelot_du_Lac
He's dead, Jim
Linus_Torvalds battles Bill_Gates
Linus_Torvalds defeats Bill_Gates
Bill_Gates battles Lancelot_du_Lac
Oh, NO!  They're both dead!  Yuck!
Status
======
Nobles:
King_Arthur has an army of 1
        Merlin: 0
```

```
Lancelot_du_Lac has an army of 2
        Conan: 0
        Hercules: 0
Jim has an army of 1
        Spock: 15
Linus_Torvalds has an army of 1
        Xena: 12
Bill_Gates has an army of 1
        Hulk: 0
Unemployed Warriors:
Tarzan: 10
Status
======
Nobles:
NONE
Unemployed Warriors:
NONE
```

Homework 6

Focus

- Cyclic Association
- Operator overloading
- Separate compilation
- Namespace

Problem:

We are revisiting the world of Nobles and Warriors (hoping you aren't sick of it...), now adding one new feature: not only can warriors be fired, they can also runaway!!! Of course, when a warrior runs away, he has to inform his noble. Which means the warrior has to know who hired him and be able to communicate with him. Which means, obviously, that he will need a pointer to his noble. Good thing we learned about forward class declarations!

Of course your Noble and Warrior classes should have all the same functionality that they had in the prior exercises, e.g. nobles can fire warriors.

Take this opportunity to clean up any "sloppy" code from before. For example, your battle method should not have redundant code. Make good use of small methods.

You will provide two solutions:

- The first is a single file, hw06-single.cpp, that solves the problem of the cyclic association and leaves no method or function definitions inside the class definition, just the prototypes and the field definitions.
- The second splits the first into separate files for separate compilation.
    - You may not have covered how to do this yet. You will in plenty of time to finish the assignment. Meanwhile, get to work on the first part above!
    - None of the code should change between the first and second implementations

- There should be separate header and implementation files for each class.
- For the separate compilation solution, place the code in the WarriorCraft namespace.

Homework 7

Focus

- Inheritance

Problem:

We want to again extend our game of Warriors and Nobles

It turns out that there is more than one type of Noble. And in fact Warriors aren't the only people they can hire to do their fighting. There is magic in the land! Life (and death) are otherwise fairly similar. Nobles are still the only ones who go around declaring war upon each other.

Nobles

Nobles come in two varieties with rather fancy sounding titles:

- Those Who are Lords of the Land
- Those With Strength to Fight

Those Who are Lords of the Land

Those Who are Lords of the Land have no strength of their own but are able to fight using the strength and skill of their army.

Thus these are the people whom we knew as Nobles in past assignments.

A Lord's army consist of Protectors. When a Lord goes into battle with another Noble, each of the Lord's Protectors will *defend* him in a way described below, under Protectors.

The Lord will win or lose depending on whether the total combined strength of his army is more or less than the strength of the opposing Noble. The strengths of each of his Protectors will be effected by the battle in the same manner as in prior assignments with the Noble's army.

The idea of "defending" is new to this assignment.

Lords can both hire and fire Protectors. When a Protector is fired, the others in the army close the gap between them, so that the Protectors maintain the same order they had before in the army. (Wouldn't want warriors having to run all about just because one of their comrades gets fired.)

Those With Strength to Fight

This type of Noble is rather different from those we have encountered before. They actually do their own fighting!!!

When doing battle, one with strength to fight will yell "UGH!!!", this being the battle cry handed down to them from the famous Barbarian, Conan. They hope that this will help them in defending themselves.

Furthermore, those With Strength to Fight fight using *only their own* strength. They do not hire anyone to fight for them and therefore do not have an army. They are born with a certain strength and they have no hope, neither through magic nor excessive exercise, to ever again increase their strength. Alas, our poor fighters will eventually have no strength left with which to fight and thus they shall meet their final demise.

Protectors

Who are these Protectors that defend Lords to the death?

- they are not Nobles!
- they are entities for hire with strength to defend. The amount of their strength set at birth.
- they are entities for hire that have names handed down from times of yore such as "QuessTar" and "VerTraahn", sacred names given at birth.

[**Clarification:** Hm, there's nothing you have to do about making sure that the names are spelled weirdly or any such. That's just there to make the story line sound more exciting.]

Lords approach Protectors to attempt to engage the service of the Protector. A Lord asks of the Protector if they are at present hired to serve another Lord and if the Protector states that he is, no transaction can take place. However if the transaction can be made, it is - and the Protector is, from that moment onward, in the service of the Lord as defender.

[**Clarification:** All this so-called dialog simply comes down to is the Lord trying to hire the Protector and succeeding if it's possible.]

In this land there are two kinds of Protectors:

- Wizards
- Warriors

They differ in their ways of defending!

This is the topic mentioned above that is new for this assigment in terms of how individual Protectors fight. In the past the only thing that mattered about one of our warrior's was their strength. Now they acutally *do* something. Too bad we don't also have animation!

Wizards state "POOF". It is such a hard job to control the strength expended with magic!

There are, further, two kinds of Warriors whose strength is spent in much more known ways:

- Archers
    - who defend by stating "TWANG! <*archer's name*> says: Take that in the name of my lord, _____" (whence he shouts the name of the lord he is sworn to defend)
- Swordsmen

    - who defend by stating "CLANG! <*swordsman's name*>says: Take that in the name of my lord, _____" (whence he shouts the name of the lord he is sworn to defend)

Again, coders beware that your code do rightly enforce all these things about a Protector, be he Wizard, Archer or Swordsman.

Loss of Strength

Each entity with strength loses it in the same manor as described in prior assignments.

Death

It's a sad topic, but one we do have to address.

- People die when they lose a battle, whether they are a Noble or a Protector.
- Lords who are dead are in no position to hire anyone. Any attempt by a dead Lord to hire someone will simple fail and the Protector will remain unhired.
- Similarly dead Protectors cannot be hired. Any attempt to hire the dead simple fails.
- However curiously, as has been seen before, Nobles can declare battle even though they are dead.
- A Protector who is dead, however, cannot fight and so will not have anything to say, even if his Lord does go into battle again.

A sample test file

```
/*  Your classes go here */

int main() {
    Lord sam("Sam");
    Archer samantha("Samantha", 200);
    sam.hires(samantha);
    Lord joe("Joe");
    PersonWithStrengthToFight randy("Randolf the Elder", 250);
    joe.battle(randy);
    joe.battle(sam);
    Lord janet("Janet");
    Swordsman hardy("TuckTuckTheHardy", 100);
    Swordsman stout("TuckTuckTheStout", 80);
    janet.hires(hardy);
    janet.hires(stout);
    PersonWithStrengthToFight barclay("Barclay the Bold", 300);
    janet.battle(barclay);
    janet.hires(samantha);
    Archer pethora("Pethora", 50);
    Archer thora("Thorapleth", 60);
    Wizard merlin("Merlin", 150);
    janet.hires(pethora);
    janet.hires(thora);
    sam.hires(merlin);
```

```
    janet.battle(barclay);
    sam.battle(barclay);
    joe.battle(barclay);
}
```

My output for the above test file is below.

```
Joe battles Randolf the Elder
Ugh!
Randolf the Elder defeats Joe
Joe battles Sam
TWANG!  Samantha says: Take that in the name of my lord, Sam
He's dead Sam
Janet battles Barclay the Bold
CLANG!  TuckTuckTheHardy says: Take that in the name of my lord, Janet
CLANG!  TuckTuckTheStout says: Take that in the name of my lord, Janet
Ugh!
Barclay the Bold defeats Janet
Janet battles Barclay the Bold
Ugh!
He's dead Barclay the Bold
Sam battles Barclay the Bold
TWANG!  Samantha says: Take that in the name of my lord, Sam
POOF!
Ugh!
Sam defeats Barclay the Bold
Joe battles Barclay the Bold
Oh, NO!  They're both dead!  Yuck!
```