Assignment 2: Refactoring and Adding Features to “Battle Game”

Spec:

Last time, you worked on porting “Battle Game” from Python to C++. This week we are going to *refactor* the game so that it utilizes Arrays, Functions, and Structs.

Make a copy of assignment 1 and put it in an “assignment 2” folder. Review the changes made in Battle Game v2 and make the appropriate changes in the C++ version.

Equipment Information – Arrays

Originally the Player and Enemy were given random stats. This time around, we'll have 3 pieces of equipment to choose from, which will give them predetermined stats: Knight's Pack, Tank's Pack, and Rogue's Pack.

Slight cleanup – Functions

We'll be delegating some parts of the program to various functions – both displaying menus and calculating values.

Equipment and Player objects – Simple Structs

This time around we're going to encapsulate the equipment information and player information into simple structs.

Dates:

Assigned:	2013-06-20	Thursday
Due:	2013-06-27	Thursday at 11:59 pm

Make sure that your source code is *committed* and *synced* to your student GitHub account. You do not need to turn in the code via Blackboard or anything else.

Assignment Tips

1. Don't try to change everything all at once! Add one thing at a time, commit that change, and then work on the next. It is much easier to write and debug if you work ***incrementally***.
2. Test often – run the program after each "feature" addition.
3. **Don't erase your whole program and start over!** Being able to refactor and add features to *existing code* is an important skill for a software developer!
4. Think about functions and objects in terms of **delegating tasks**; "Game begins... then we display the title menu... then we ask the user for their choice... then we show the equipment menu..." For example, this could then become DisplayTitleMenu(), GetUserChoice(), and ShowEquipmentMenu().
5. If you can't finish a feature, make sure your program still runs and works beyond that one feature/refactoring.

Step-by-Step: Suggested path on adding new features

1. Refactor: Moving Character Stats into a Struct

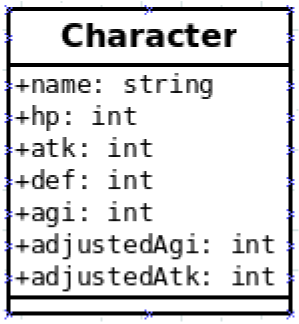
Create a new file in your project: **Character.h**

In the current program, we're creating several variables to hold the Player and Enemy's stats:

```
int playerHP = 20;
int playerAtk = rand() % 3 + 5;
int playerDef = rand() % 3 + 5;
int playerAgi = rand() % 3 + 5;
```

Really, this data should be grouped together into one object – we can create a struct that, instead, stores "hp", "atk", "def", "agi" within it. Additionally, the Player and Enemy both have *the same stats*, so we <u>*do not need to create separate structs for each!*</u> Simply create a generic struct, like "Character", give it the appropriate variables, and then create ***instantiations*** of that struct (ie, declare two variables of type Character).

Create a **struct named Character within Character.h.** Here's a sample of what your struct should have:



Then within main.cpp (or whichever source file is storing your main() function), make sure to #include "Character.h".

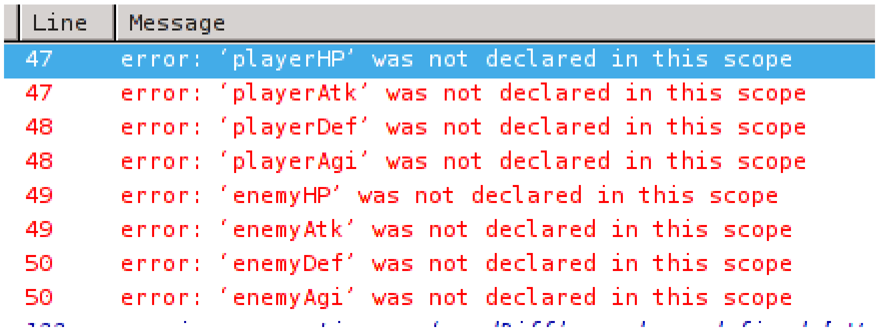
Note: Don't forget the #ifndef commands in the .h file, and remember that if our struct uses string objects, we need to #include <string> and use the appropriate namespace!

In main(), we'll replace the initial lines of code where we're creating those Player and Enemy variables with our struct:

Old	**New (Sample)**

`int playerHP = 20;`
`int playerAtk = rand() % 3 + 5;`
`int playerDef = rand() % 3 + 5;`
`int playerAgi = rand() % 3 + 5;`
	`Character player;`
	`player.name = "Player";`
	`player.hp = 20;`
	`player.atk = rand() % 3 + 5;`
	
	And so on... (Continue for the rest of *player* and then create an *enemy* instance of Character)

Once you replace the *playerHP, playerAtk, enemyHP, … etc.* variables, if you build your program you will get compile errors.



```
Line  Message
47    error: 'playerHP' was not declared in this scope
47    error: 'playerAtk' was not declared in this scope
48    error: 'playerDef' was not declared in this scope
48    error: 'playerAgi' was not declared in this scope
49    error: 'enemyHP' was not declared in this scope
49    error: 'enemyAtk' was not declared in this scope
50    error: 'enemyDef' was not declared in this scope
50    error: 'enemyAgi' was not declared in this scope
```

The next step of refactoring is to fix these locations – for example, where it is using "`playerHP`", change it to use "`player.hp`" instead.

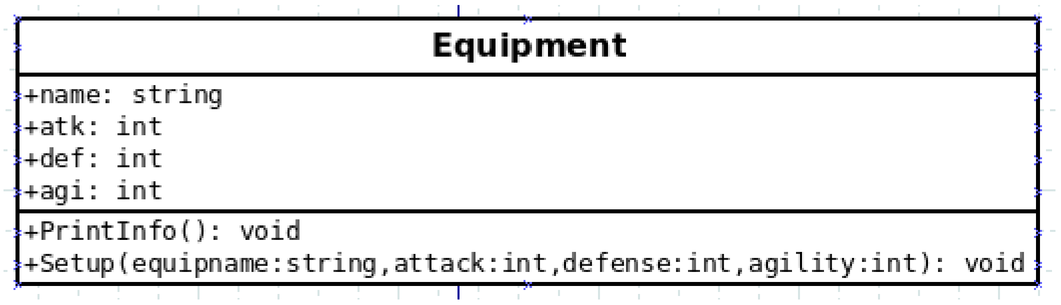
Replace everywhere you can find, continually building, until you have no more errors and the program again originally works like it used to.

At this point, nothing else in the program should have changed – Only replacing the Atk/Def/Agi/etc. Variables with our Struct version. Once everything builds and runs again – congratulations, you have just done your first refactoring!

2. Feature: Equipment

Next, we're going to add a feature to our program – equipment. Currently, characters' stats are randomly generated, so winning the game is somewhat “dumb luck”.

With this feature, we're going to add several different types of equipment the user and computer can choose from. Each equipment has its own set of “Atk”, “Def”, and “Agi” stats that affect the user's own Atk, Def, and Agi stats. It will also have a “PrintInfo()” function.



1. Create **Equipment.h** and **Equipment.cpp**.
2. Create the Struct declaration in Equipment.h
3. Create the Equipment::PrintInfo() function definition within Equipment.cpp.
4. Create the Equipment::Setup(…) function definition within Equipment.cpp

Inside the **PrintInfo()** function, you will output the equipment's name, atk, def, and agi. Remember that you need to include iostream and use the namespace.

Inside the **Setup(...)** function, we have several parameters passed in:

- “equipname”, a string data-type
- “attack”, an int data-type
- “defense”, an int data-type
- “agility”, an int data-type

Our Equipment struct has variables name, atk, def, and agi. Assign these struct variables to the values passed in as parameters. (name is equipname, atk is attack, and so on).

Back in main.cpp, make sure to `#include "Equipment.h"`.

Inside of main(), somewhere around where we're setting up Player and Enemy, create an array of Equipment, with a size of 3.

Then, we need to give all 3 pieces of equipment some data. We will use the Setup function:

```
Equipment equipment[3];
equipment[0].Setup( "Knight's Pack", 8, 4, 3 );
equipment[1].Setup( "Tank's Pack",   4, 8, 3 );
equipment[2].Setup( "Rogue's Pack",  4, 3, 8 );
```

Build your program, and see if it runs.
If there are any compile errors, fix them before proceeding.

Now that we've initialized types of equipment, we want to have the user and computer select an equipment to use. We will create a new menu that shows each equipment's information (via the Equipment's PrintInfo() function), and then sets the player's stats based on the equipment they chose.

So quickly, let's look at the states our program goes through:

- Title screen – Display title screen, display game instructions, and ask if the user is ready to begin. Generate random stats for Player and Enemy.
- Main game loop – Each time through the loop counts as one round. Each round, the player chooses the type of attack, then the Enemy randomly chooses theirs. Then, the program steps through each character's attack and updates their HP.
- Game over – Once the player or enemy have "died", the game exits the loop and either a "congratulations" or "failure" message is displayed.

We need to add another "State", where the Player and Enemy get to choose their equipment from the 3 options. This will go after the title-screen and before the main game loop, and will only occur once.

Additionally, we no longer need to randomly generate the Player and Enemy stats. Keep the lines where we set the player and enemy names and HP, but erase where we set .atk, .def, and .agi to random values.

Somewhere before the "Are you ready to begin?" prompt in the game, we are going to add the equipment choice menu.

The equipment choice menu's anatomy is simple: use a for-loop to iterate from 0 to 3, and output each equipment's information.

Then, prompt the user to choose one, get their choice (must be 0, 1, or 2), and copy the equipment's stats to the player's stats.

For the enemy, have the program randomly generate a number between 0 and 2 and then copy the equipment's stats to the enemy's stats.

```
# Display all equipment
for i in range( 0, 3 ):
    print( "Choice " + str( i ) )
    equipment[i].PrintInfo()

choice = input( "Which equipment? " )
choice = int( choice ) # Do not need to cast in C++

player.atk = equipment[ choice ].atk
# And so on for the other stats
```

Afterwards, we should keep printing out the player and enemy's stats, then ask if the user is ready to play the game.

3. Cleaning up main()

Finally, we should spend some time refactoring our main() function. Currently, *my* main function is 180 lines long! We're handing all of our game logic within main itself, when really separate functionality should be handled in separate areas.

The first step is to think of our program in terms of *steps*, and then thinking about which variables are shared between these steps:

1. Game Beginning
   1. Start game, initialize variables
   2. Display title screen and instructions
   3. Display equipment options
   4. Get user choice and enemy choice
   5. Copy equipment stat values to player and enemy's stats
   6. Display each character's stats
   7. Ask if they're ready to begin the game.
2. Round Loop
   1. Display the Round menu, which includes the player and enemy stats, and possible attack choices: Light, standard, and heavy.
   2. Get the user and enemy's choice of attack
   3. Calculate adjusted Attack and Agility ratings.
   4. Step-through series of messages as the battle plays out – Player attacks Enemy, Enemy attacks Player!
   5. Adjust enemy and player HP
   6. Check to see if either player or enemy has died
      1. If so, set the “Game over” condition (boolean)
      2. If not, don't change the game over variable and loop again
3. Game Over
   1. Check to see who has 0 HP.
      1. If the enemy has 0 HP, congratulate the player
      2. If the player has 0 HP, hassle them over failing
4. End the program.

Each and every one of these steps are currently handled inside main(). This is not well-designed

We could separate most of these steps into their own functions. Or, if we could identify a commonality between a series of steps, we could encapsulate them in a class that contains several functions.

Let's just worry about extracting some space-hogs into their own functions for now.

void DisplayStats(Character player, Character enemy);

The DisplayStats function handles outputting the Player and Enemy's name, atk, agi, def, and hp. All it does is handle cout-ing data and does not return any values.

Create a function within the main.cpp file (or wherever main() is) called DisplayStats. It will have the player and enemy as parameters.

With the player and enemy objects, it will output name, atk, agi, def, and hp.

That's all this function will do.

Then, within main(), make sure to replace two areas where stats are displayed: Right after the equipment select, and at the beginning of a round.

Character AdjustStats(Character character, int attackChoice);

The AdjustStats function will be called after the player and enemy have chosen their attack type. Instead of having an if-statement block to decide what to set their adjusted stats to based on the player/enemy choice, we will call the AdjustStats function to handle adjusting a character stats instead.

The function's return value is the character whose stats have been updated.

Create a function called AdjustStats. Its parameter is a Character object and the attackChoice and it will return a Character object. This is made *generic*, so that either Player or Enemy could be passed into it.

Within the function, move the if statements from within the round that adjust the player and enemy's atk and agi based on their choice:

```
if ( playerChoice == 1 )
{
     player.adjustedAgi = player.agi + 2;
     player.adjustedAtk = player.atk - 1;
}
else if ( playerChoice == 3 )
{
     player.adjustedAtk = player.atk + 2;
     player.adjustedAgi = player.agi - 1;
}
```

This logic will be moved into the AdjustStats function, except it is not specific to player.

Since our parameter is named "character", you will add or subtract the adjusted stats for that object.

Don't forget to return the "character" at the end of the function.

Then, within the game loop you will erase that section of code (that adjusts the player/enemy stats based on their choice), and instead pass in that choice variable to the AdjustStats function:

```
cin >> playerChoice;
enemyChoice = rand() % 3 + 1;

player = AdjustStats( player, playerChoice );
enemy = AdjustStats( enemy, enemyChoice );
```

Build and run the program, and make sure it's still working as it should.

That's all the mandatory refactoring for this assignment. If you wish to refactor or restructure the program for extra credit, make sure to commit and push this final "version" (make a note in the commit message that it's the "finished" version), and then continue working and committing.