Template Functions, Function Overloading, Structs

Sannihitha Muppidi

October 11th, 2016

1 Problem Statement

As the title of this lab implies, the lab will cover template functions, function overloading, and structs (also a few other miscellaneous topics). This will be a longer lab, so please use time wisely. I recommend reading through the entirety of this lab, especially to the hints.

The scope of this lab is to create a simplistic attack-and-defend game. Two battleships start with the same health, but continuously attack each other until either one gets sunk or one flees from taking too much damage.

The implementation is laid out as follows.

Required Files

There will be *five* files to submit, namely

1. `main.cpp`
2. `battleship.h`
3. `functions.cpp`
4. `functions.h`
5. `templates.hpp`

And an optional constants file.

Required Structs

There will be one struct to be implemented, called `BattleShip`. It will have the following variables:

- A health variable of type `int`. This holds the health points (hp) of the ship.
- A defense variable of type `short`. It will be used when calculating relative damage.
- An attack variable of type `short`. It will also be used to calculate relative damage.

Required Functions

The following are all the *required* functions that must be implemented — this does not include any maintenance functions you so choose to have.

- `T randomNumber(const T from, const T to)` A **templated** random number generator (of typename `T`) that returns a random number `from`...`to` (inclusive). This **does not** work for floating point numbers, that is okay.
- `readyTheShip(BattleShip & ship, const int shipNumber)` This function prepares the ship for battle. Note that the
  1. Ask for input on what the name of the ship is.
  2. Assign health to a value of 100.
  3. Assign defense a *random* number 1 . . . 5. (1 through 5)
  4. Assign attack a *random* number 15 . . . 20. (1 through 5)
- `void attack(BattleShip & attacker, BattleShip & attackee)` The attackee ship loses health depending on the attack value. The attack value is a *random* value `attacker.defense`...`attackee.attack`. Output the damage.
- `bool attemptToFlee()` The flee is completely up to chance. At first the percentage chance is 15%, but with every successive attempt to flee, the change goes down a 1% until it's fixed at 1%.

- `void winner(const string winner)` Outputs a single winner.
- `void winner()` Outputs that there was a fleeing ship, no one wins!
- `int main()` Because there are several ways to play a game like this, there is provided pseudocode below. If you find a clever or cool different way to play, ask any of the student TAs or the TA if you are allowed to implement it.

```
ready ship one and two

while no one fleed and no ship dies:
    ship two attacks ship one
    ship one attacks ship two

    if the health of ship one is < 20:
        ship one attempts to flee

    if the health of ship two is < 20:
        ship two attempts to flee

if someone flee,
    display flee message
else
    present the winner
```

Note that we do not consider ties, if two lose at the same time, arbitrarily choose the loser (I recommend choosing Price, if possible).

2 Submission

When using `cssubmit`, please use the following sample input and verify your solution with the sample output.

Sample Input

```
Jarus
Price
```

Sample Output

```
Jarus dealt 9 damage to Price
Price dealt 10 damage to Jarus
Jarus dealt 11 damage to Price
Price dealt 5 damage to Jarus
Jarus dealt 12 damage to Price
Price dealt 16 damage to Jarus
Jarus dealt 6 damage to Price
Price dealt 16 damage to Jarus
Jarus dealt 14 damage to Price
Price dealt 17 damage to Jarus
Jarus dealt 11 damage to Price
Price dealt 8 damage to Jarus
Jarus dealt 15 damage to Price
Price dealt 15 damage to Jarus
Jarus dealt 13 damage to Price
Price dealt 8 damage to Jarus
Jarus dealt 11 damage to Price
Price dealt 17 damage to Jarus

Jarus won!
```

3 Hints

- Compile early and compile often.
- If you don't know/aren't quite sure how to generate a random number $x_0 \ldots x_n$, ask one of the TAs for a full explanation.
  - Make sure your random number generator is *inclusive*.
- There is a bare minimum of 6 constants in this assignment.