2E3 Lab 4

In this lab you will use object oriented design and programming techniques to design and implement a new class in C++ to encapsulate the behaviour of a role playing game non-player character (RPG NPC).

Your NPC has the following traits (which will need to be stored in class data members/attributes):

- Strength (1 to 20)
- Skill (1 to 5)
- Alive (true or false)

Your NPC has the following abilities (which will need to be implemented as methods):

- print, which prints the NPC's vital statistics (strength, skill, alive) to the screen (via `cout`).
- attack, which rolls a 6-sided dice, multiplies the result by the NPC's skill and returns the result, which is the NPC's attack strength.
- defend, which takes an attack strength as a parameter. This method rolls a 6-sided dice and multiplies the result of the dice roll by the NPC's skill to obtain the strength of the defence. The strength of the defence is subtracted from the supplied attack strength. If the strength of defence was larger than the strength of the attack (i.e. if the result is a negative number), no damage to character strength was done. If the strength of attack was larger, i.e. if the result is a positive number, it is subtracted from the NPC's strength. If the NPC's strength reaches zero, then the NPC is dead.
- isAlive, which returns whether the NPC is alive or dead.

1. Design a class, called `NPC`, which encapsulates all the behaviour and NPC state outlined above. Provide a full class definition (.h file) and the implementations (in .cpp) for each of the methods. You should supply a default constructor, and an additional constructor that allows all the parameters of the NPC to be specified.
2. Design a class, called Team, which manages a linked list of Characters. The class Team should have the following public methods:
   - void add(NPC *npc), which adds a new NPC to the list (e.g. at the end/tail)
   - int size(), which returns the number of characters currently in the list
   - NPC* at(int i), which returns a pointer to the NPC at index i, when i is between 1 and SIZE, included, NULL otherwise.

You should provide a full class definition (.h file) and the implementations (in .cpp) for these methods and all you need to support them.

3. Use your classes to create a simple application that (I) asks the user for the size (N) of the teams, (ii) creates 2 teams containing N NPCs each (2 different teams of NPCs). Each NPC should be initialized with random initial strength and skill.
4. Implement the game in which NPCs from opposing teams with corresponding array indexes attack each other (i.e., homeNPC.at(i) attacks awayNPC.at(i) etc). The fight will have two rounds: allow home team to attack first in the first round,

and then in the second round the away team to attack the home team. (Note that only alive NPCs can attack so before attacking check if NPC is still alive).

5. At the end of the attacks determine the winner by counting how many NPCs are alive on each time – the team with more alive NPCs is the winner. If both teams have equal number of alive NPC, it's a draw!
6. Modify the game so that one character in each team (at a random place in the team) will be a character with a special trait, i.e. its strength will never fall below 1 when attacked. This character will have to be implemented in a different class, in files Wizard.h and Wizard.cpp, without any modification to the NPC and to the Team classes.

TO GET YOU STARTED

You can use a c++ function rand() to generate random numbers. To limit the range of random numbers generated use the modulus operator.

e.g. to get a number between 0 and 5: `int roll = rand()%6;`

(add 1 to this number to get a number between 1 and 6, i.e. to simulate a dice roll)

However, For a given *seed* (starting value), the sequence of numbers that **rand()** returns will always be the same, so if you run your program several times, randomly generated character attributes will be the same in each run. To make sure your random number generator is seeded with a different value each time, seed it with the current time, by adding the following line at the start of your program.

```
srand(time(0));
```

Note that you will need to include #include <ctime> in order to use the function time()

HINTS:

Use as much as possible the code from previous labs dedicated to linked lists.

The marks for this lab will be assigned as follows:
0 = did not attend the lab session
1 = attended the lab session but have no or very little working code or **THE CODE DOES NOT COMPILE. Your code MUST compile in order to get a mark higher than 1.**
1 = header file specification for class NPC
1 = header file specification for class Team
2 = correct implementation of NPC class
2 = correct implementation of Team class
2 = correct fight implementation
1 = addition of special character without modifications to other classes
---
Max 10 points

The program needs to be DEMONSTRATED for marks to one of the demonstrators before the end of your lab session on November 18th/19th AS WELL AS SUBMITTED ONLINE AS LAB NUMBER 4 by Friday Nov 20th 5pm through Blackboard
(submit all .h and .cpp files).