Excercise set week five

Feedback on these excercises is very much appreciated. Send mail to one of the lecturers or to Aryan at aryannm@gmail.com

Solutions are available in file `solutions.md`.

Problem X: (Repetition)

1. Create a class `Student` that has data members `studentnumber` and `name`. Decide if they should be string/int etc.
2. Create a class `SchoolClass` that has a list of `Student` has datamember, decide if this list should be an array or `std::vector`.
3. Creating an empty class with 2 datamembers and maybe a constructor is somewhat boring. Extend both classes so they support setting and getting values (also known as encapsulation, refer to google.com or solution).
4. Now that you have both classes fleshed out, which of the member methods should be using keyword `const` in the function signature and which shouldn't? Why? Why not?

Problem One - Static

Refer to problem X.

Suppose we are interested in knowing how many Students have been created and we want this as a counter inside the `Student` class, one way to do this would be to add a `static int count` variable inside the `Student` class and increment the value by one every time a student is created in the constructor(s) of `student`.

1. Make changes to `Student` to add this functionality.
2. Create a static function inside the student class for getting this number.

Problem Two - Header files & inlining

Refer to problemset of week 5 and the solution there if you get stuck.

1. Seperate the classes that you just created into `.hpp` and `.cpp` files, a header and implementation file respectively.
2. What do you think is the point of this?
3. Some of the member functions/methods are good candidates for inlining using the keyword `inline`, do the changes necessary to make them inline function.
4. Why do we need or want inlining?

Problem Three - Function Overloading

Refer to lecture notes for details.

1. Create a function called `void foo(int a)` that prints out the value it takes with the output "Hello my int value is:"
2. Now create overloaded functions that takes `char c`, `std::string s` and `long l` as parameters and print similar output, with type and value.
3. What do you think the point of overloading a function is?

Problem Four - Inheritance

Suppose you are creating a game and instead of creating new classes all the time, eg AggressiveMonster, PassiveMonster, FriendlyMonster etc you find out that you can save yourself a lot of typing and structure your code better if you use the concept of inheritance.

If you can, try and make the classes into seperate `hpp` and `cpp` files.

1. Create a class called `Player` with members `int x`, `int y`, `int hp`, `std::string name`, create appropiate setters and getters.
2. Create a class `Monster` that has datamembers `int x`, `int y`, `int hp`, `int attacktimer`, create appropiate setters and getters.
3. Hmm it seems a `Monster` and a `Player` share a lot of common things, create a class called `Creature` with datamembers `int x`, `int y`, `int hp`, `std::string name` with appropiate setters and getters and use inheritance to make your `Monster` and `Player` classes to inherit from this `Creature`.
4. To your `Monster` class add a member method called `void seek(Player p)`. Now create a `PassiveMonster` class and a `AggressiveMonster` class, how would you use function overriding to change `seek` for `PassiveMonster` and `AggressiveMonster`? You might have to refer the C++ book on virtual functions (or use google).
5. (Extra) Create a class `Game` with two datamembers `std::vector<Monster*> creatures` and `Player player`. Create a player and a Monster and add them to the game (maybe Game should have appropiate setters and getters?). This is another very powerful feature of inheritance, any speciality monster also happens to be a `Monster`.

Notice that the vector contains `Monster*` and NOT `Monster`, the reason for this is calld slicing. Further research in the book or google (or send me a mail).