# COMP 215 Algorithms

## Lab #1

The goal of this lab is essentially to get you to write C++ code again, shake off the rust that may have accumulated since the last time you had to write code. I will ask you to implement a relatively simple class for a `NestingDoll` with a few methods, and a main function to test it. As you complete the exercises, identify the elements you have trouble with, if many people have the same, I may do a quick review of those topics.
Your project should contain three files:

- a file `NestingDoll.h` that contains only the class declaration and the declaration of internal variables and methods, all of it between include guards (read the Wikipedia page on **Include Guards** if you need a refresher)
- a file `NestingDoll.cpp` that contains the implementation of all the methods from `NestingDoll.h`
- a file `main.cpp` that contains the code you wrote to test the methods. Leave all your testing code there (you can comment it out, but do not erase it), I will have a quick look at it to see how you tested the methods.

The class should contain the following *private* variables:

- an integer `size`,
- a string `color`,
- a pointer to a `NestingDoll` called `insideDoll` (which, naturally, points to the doll that is immediately inside the current doll).

The class should contain the following constructors and destructor:

- a constructor that takes 3 input parameters, an integer, a string and a pointer to a `NestingDoll`, and do the obvious thing. The third parameter should have a default value of `nullptr` so that it can be omitted when calling the constructor. If the pointer to the `NestingDoll` is not `nullptr`, then you should test if the size of the input doll is smaller than the size of the doll currently being created. If it is, you can put the input doll 'inside' the doll you are creating. If the input doll is of equal size or larger, then you should print an error message and the pointer of the doll you are creating should be left as `nullptr`.
- a copy constructor that makes a copy of the input doll, as well as a copy of all the dolls inside of it (ALL the dolls should be copied)
- a destructor that destroys all the dolls inside the doll.

The class should contain getter methods for the size and color.

Add also the following methods that change or reveal the internal state of the object:

- a method `isEmpty` that returns `true` if there is no doll inside the current object, and `false` otherwise.
- a method `remove` that removes the internal doll by setting the internal pointer to `nullptr` and returns the doll that used to be inside. The method should return `nullptr` if the current object was empty.
- a method `putInside` that takes a pointer to a `NestingDoll` as input parameter and does the following:
  - if the current doll is not empty, print an appropriate error message and do nothing,
  - if the current doll does not have a larger size than the input doll, print an appropriate error message and do nothing
  - otherwise, put the input doll inside of the current doll.

Finally, overload the following operators:

- `operator[]` takes an integer as input parameter and returns a pointer to a `NestingDoll` as follows:
  - if the integer is negative, return `nullptr`,
  - if the integer is greater than the number of dolls inside the current doll, return `nullptr`,
  - if the integer is zero, return a pointer to the current doll,
  - otherwise, 'open up' the dolls a number of times equal to the input parameter, and return a pointer to the doll you find inside
- write `operator<` so that if $A$ and $B$ are dolls, then $A < B$ is true if the size of $B$ is greater than the size of $A$ AND $B$ is empty. (basically, $A < B$ if it would be possible to put A inside of B)
- write `operator>` so that it is consistent with `operator<`.

Once you are done with everything, submit a `.zip` file containing `NestingDoll.h`, `NestingDoll.cpp` and `main.cpp` on onCourse.