Foreword

In order to do this lab you must have access to the school's virtual machine (see wiki on Moodle), your code from the previous lab.

In this exercise, you will cover the following topics:

- Overloading operators;
- Writing an object function;
- Using lambda functions;
- Redraw trees in console ***[BONUS]***.

Advice

- Use `man` an especially `man 3` for the development pages ;
- The website `https://en.cppreference.com` is your most valuable friend for the Teaching Unit;

Part I
Implement a move constructor

Instruction

Create a move constructor for the `Pine` and `Oak` classes;

Question 1

- How can you check which constructor is called ?
- Use `std::move()` to force the usage of this constructor.

Part II
Create a Lumberjack class

1 Update your trees

At the moment the `width` and `height` attributes of the `Pine` and `Oak` classes are linked to the size of the `image` to represent the specific tree.

Instruction

- For this lab, we will add a `double size` attribute to the `Tree` abstract class in order to represent the height of the real tree and not of its image in the console[a].
- Update the constructors for the `Pine` and `Oak` classes so that the `size` attribute is assigned randomly[b]
- Update the `info()` methods to print out the size of the trees.

[a] In order to keep it simple, this attribute can be declared public at first
[b] Try to use the `alea()` template in `utils.h`.

Question 1

- Check that each tree has a different size ?

2 Creating the Lumberjack

We will now create a `Lumberjack` class that will be in charge of cutting down trees that are above a certain size.

Instruction

Create a simple `Lumberjack` class with two attributes :

- A `double` to represent the cutting height of the lumberjack;
- An `int` to count the number of trees that have been cut.

Implement a specific constructor to set the cutting height and to initialize the counter.

Check that you can create a `Lumberjack` in your main program

The purpose of our `Lumberjack` is to cut trees that are above a certain height. What we want is not to delete `Tree` objects but to modify the `size` attribute of the `Tree`. To do so, we need to implement the "()" operator for the `Lumberjack`.

Instruction

Overload the "()" operator for the `Lumberjack` class so that it modifies the `size` of a `Tree` parameter if its size is over the limit.

The signature should look like this :

```
void operator() (Tree& t);
```

Test your code with the following program :

```
#include <iostream>
#include "pine.h"
#include "lumberjack.h"


int main(int argc, char** argv) {
  std::cout << "Launching the main program - copy constructor" << std::endl;
  //Pine creation
  Pine p1;
  Lumberjack bob (3.0);

  p1.info();

  bob(p1);

  p1.info();

}
```

Question 2

- The signature of the operator takes a reference to a `Tree`, why is it working with the `Pine`?
- What is the purpose of using our `Lumberjack` instead of just changing the size of the Tree ?

3 Replacing the lumberjack by a lambda function

Instruction

Replace our `Lumberjack` class by a lambda function.

Part III

BONUS - Visualizing the lumberjack work

Even if the `Lumberjack` or the lambda function is called and the trees are cut to the parametrized size, the image does not change if we call the `draw()` method. In this bonus section we want to modify the `image` of each tree in order to visualize the process.

Instruction

- Compute the ratio between the original size and the new size;
- Apply this ratio to the `height` attribute minus one[a];
- Remove part of the image that have been cut down to visualize that the tree has been cut down.

[a]As you can see on the console the tree is always 4 pixels high but the trunk occupy one pixel

Instruction

- Try to create a whole forest of random trees (`Pine` and `Oak`) and call the `Lumberjack` (or lambda) on each tree of the forest.
- visualize the forest before and after the `Lumberjack` call

Appendices : Useful commands

```
$man COMMAND                 #display the manual page for the given COMMAND
$man 3 FUNCTION              #display the developer manual for the given FUNCTION
$g++                         #GNU project C and C++ compiler
$make                        #GNU make utility to maintain groups of programs
$ldd                         #print shared object dependencies
$strace                      #trace system calls and signals
$strings                     #print the sequences of printable characters in files
$gdb                         #The GNU Debugger
```