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 or the code snippet provided on Moodle to draw the images.

In this exercise, you will cover the following topics:

- Understanding memory management and its relation to the modeling of a solution;
- Declaring and using a copy constructor;
- Overloading operators;
- Draw trees on the 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;

Warning

In the sources provided on Moodle you will find some additionnal code in order to draw the image of a tree on your terminal's console. It is composed of three specific files :

- `pixel.h` which defines a `Pixel` class as well as some inline implementation
- `utils.h` which defines two helper functions among which the `buildImage()` one to create an image of a `Tree`
- `utils.cpp` provide the actual code for the `buildImage()` function and needs to be compiled and linked with your main program

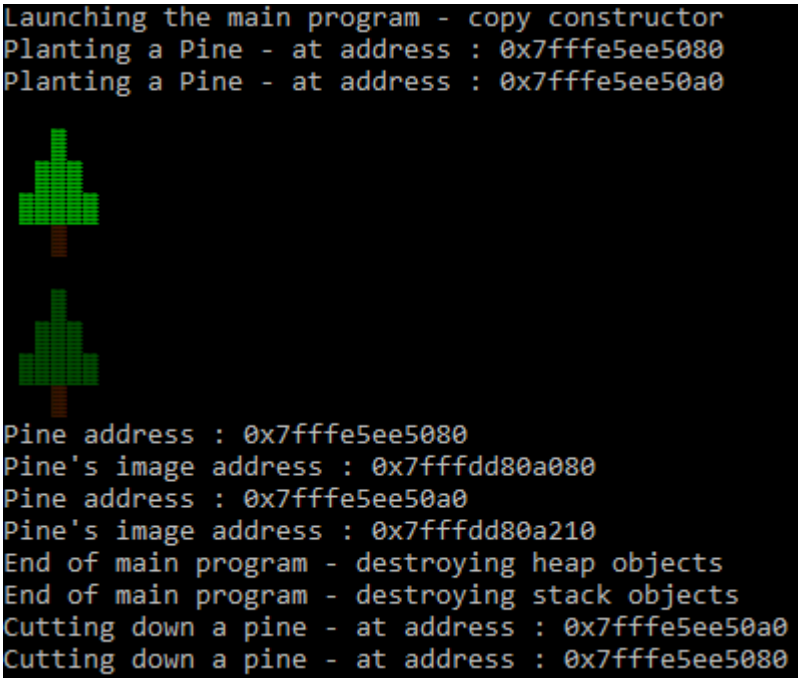

Figure 1: Two pines drawn on the terminal

Part I
A `Pine` class with an embedded image

1 First look at the constructor

With the code provided on Moodle you can draw images of trees on the terminal's console (see Fig 1). However, the provided program still has a lot of bugs that need to be corrected.

Instruction

Take a first look at the new class definition in `pine.h` and at the default constructor implementation in `pine.cpp`.

Question 1

- What are the `Pine` class attributes ?
- By just reading the `pine.h` file can you decide whether the relation between a `Pine` and an `image` (which is a pointer to a `Pixel` array) is an aggregation or a composition ?
- What is the purpose of the `Pine` constructor ? Having looked into the constructor implementation can you answer the previous question now ?

2 Creating the destructor

In the provided code, the `image` attribute is an array of `Pixel` objects allocated on the heap by the constructor. For this lab we will choose to model the relation between the `Pine` and the `image` as a composition. That means that when a `Pine` is cut down, its `image` should be removed as well[1].

Instruction

Implement the correct destructor for the `Pine` class to delete the image when the `Pine` is destroyed.

In your main program, declare a `Pine` on the stack and call both the `draw()` and `info()` methods and check that the `Pine` is drawn on the console.

[1] We could have chosen to model the relation as an aggregation to share an image between several trees (see Flyweight Design Pattern) in order to save memory for example

Part II
Copying a Pine

3 Shallow copy

The default C++ mechanism is to create a shallow copy of an object, in this section we will explore the implications of such implementation for our `Pine` class.

Instruction

Without changing the `Pine` class, create a simple program that declares two trees `p1` and `p2` (in order to see the two different colors) and create a third pine `p3` which is a copy of `p1`. See example below :

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


int main(int argc, char** argv) {
  std::cout << "Launching the main program - copy constructor" << std::endl;
  //Pine creation
  Pine p1;
  Pine p2;

  //Pine copy
  std::cout << "Attempt to copy first Pine using the Copy constructor" << std::endl;
  Pine p3(p1);

  //Pine drawing
  p1.draw();
  p2.draw();
  p3.draw();

  //Pine info

  p1.info();
  p2.info();
  p3.info();

  std::cout << "End of main program - destroying heap objects" << std::endl;

  std::cout << "End of main program - destroying stack objects" << std::endl;
  return 0;
}
```

Question 2

- Does the program compile ?
- Does the program run as intended ?
- Explain the issue ?

4 Copy constructor

The shallow copy mechanism is not adequate for our choice of the composition relation between the `Pine` and the `image`. In fact, when copying the `image` of the pine `p1`, all the compiler do is copy the address pointed by `image` in `p3`. Doing so, both `p1` and `p3` have a pointer to the same image. We want to implement a deep copy mechanism so that each `Pine` "owns" its `image`.

Instruction

Create a copy constructor with the following signature :

```
Pine(const Pine& s);
```

The idea of this constructor is to create a new `image` and copy the memory stored in the referenced `Pine` into this new `image`[a].

Compile the previous program and check that now the copy is working by looking at the `images` addresses.

[a] Check the manual page for the `memcpy()` function

Part III
Affectation operator

5 Default behavior

Instruction

In your main program, add the following code after the previous copies:

```
std::cout << "Affectation p3 = p2 " << std::endl;
p3 = p2;

//Pine drawing
p1.draw();
p2.draw();
p3.draw();

//Pine info

p1.info();
p2.info();
p3.info();
```

Question 3

- Does the program compile ?
- Does the program run as intended ?
- Explain the issue ?

6 Implementing the affectation operator

In the previous code, the equal operator behaves like the default copy mechanism and both `p2` and `p3` share a pointer to the same `image`. Again, this goes against our composition relation between the `Pine` and the `image` which is not what is expected. To solve this issue, we need to overload the default "=" operator.

Instruction

Overload the "=" operator :

```
Pine& operator=(const Pine& s);
```

The idea of this operator is :

- First check that you're not attempting to auto-assign (for ex: p3 = p3);
- Then check that the two objects have images of the same size;
- Finally, copy the memory stored in the referenced `Pine` into the second one.

Compile the previous program and check that the affectation is working by looking at the `images` addresses.

Question 4

- What can happen if the two images are not of the same size ?
- How can you make sure that the affectation work in all case ?

Instruction

Implement the solution to the previous question.

7 Copy constructor vs. Affection operator

Instruction

Change your main program to add the following line:

```
Pine p4 = p1;
```

Question 5

Which method is called : the copy constructor or the "=" operator ?

Part IV
BONUS - Create an Oak Class

Instruction

- Create an new `Oak` class that inherits from `Tree` and create your own texture image to represent it [a];
- Implement the constructor, the copy constructor and the affectation operator for the class `Oak`;
- Create a main program that specifically asks the user for the type of `Tree` she wants to draw and how many of them your program should draw;
- Run and test your program.

[a] Read the `utils.h` to help you and remember that an oak is not evergreen so you can choose a different color

Instruction

- Change the `info()` methods by an overloaded $\ll$ operator that prints the same message.

Question 6

Can you use the polymorphism with this new operator ?

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
```