UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 4 Specifications

Release Date: 19-08-2024 at 06:00

Due Date: 23-08-2024 at 23:59

Total Marks: 150

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note, UML notation for variable and function signatures are used in this assignment.

2 Overview

Operator overloading in C++ allows you to define custom behaviors for operators when applied to user-defined types, enhancing code readability and flexibility. This enhances the flexibility and readability of your code. In this practical you will simulate colours being mixed, to better your understanding of operator overloading in C++.

3 Background

The RGB colour model is an additive colour model in which the red, green and blue primary colours of light are added together in various ways to reproduce a broad array of colours. Each of the values

are an integer between 0 and 255. In this practical you will implement a Colour class and a Mixer class to simulate how the RGB values of colours are used to produce new colours.

4 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

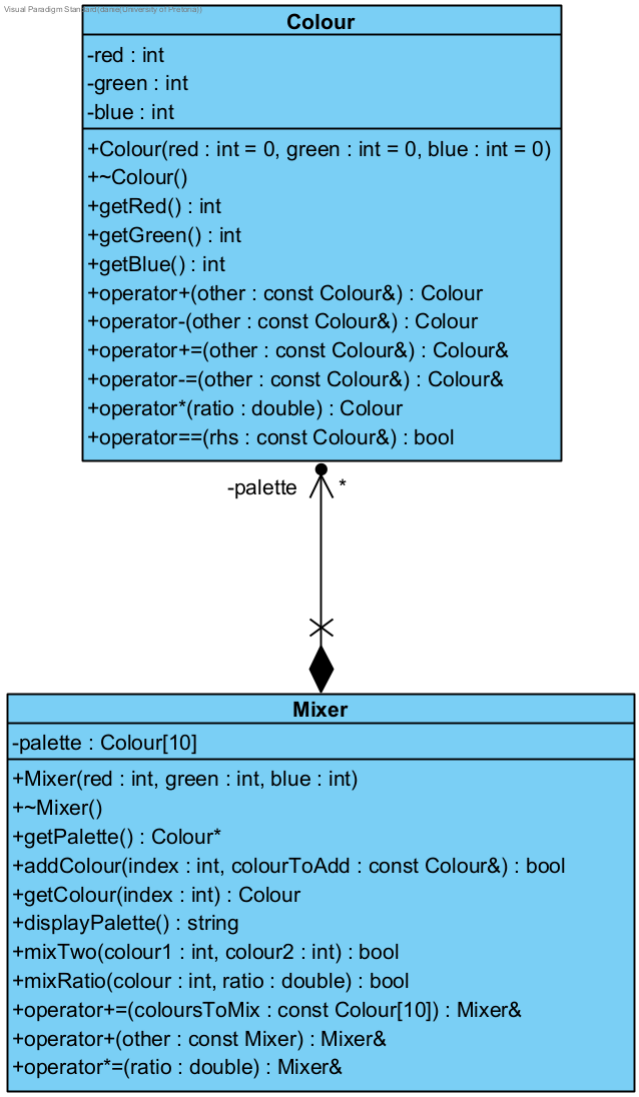

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

4.1 Colour

The colour class represents a colour with RGB values and have methods to change the RGB values.

4.1.1 Members

- -red: int
  - This member variable contains the red value of the colour.
- -green: int
  - This member variable contains the green value of the colour.
- -blue: int
  - This member variable contains the blue value of the colour.

4.1.2 Functions

- +Colour(red : int = 0, green : int = 0, blue : int = 0)
  - This is the constructor for the Colour class.
  - It should assign the passed in red, green and blue values to the corresponding member variables.
- +∼Colour()
  - This is the destructor for the Colour class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- +getRed() : int
  - This is a get method for the red member variable.
  - It should return the red member variable.
- +getGreen() : int
  - This is a get method for the green member variable.
  - It should return the green member variable.
- +getBlue() : int
  - This is a get method for the blue member variable.
  - It should return the blue member variable.
- +operator+(other : const Colour&) : Colour
  - This is a method used to combine the RGB values of two colours.

  - It should return a new colour where the RGB values are obtained using the following formula: $\frac{value+otherValue}{2}$.

- +operator-(other : const Colour&) : Colour
  - This is a method used to combine the RGB values of two colours.
  - It should return a new colour where the RGB values are obtained using the following formula: $\frac{|value-otherValue|}{2}$.

- +operator+=(other : const Colour&) : Colour&
  - This is a method used to update the RGB values of a single colour by mixing it with another colour.
  - It should add the result of the following formula to the current RGB values of the colour: $\frac{value+otherValue}{2}$.
  - It is important that if one of the RGB values exceed 255 that it should be set to 255.

- +operator-=(other : const Colour&) : Colour&
  - This is a method used to update the RGB values of a single colour by mixing it with another colour.
  - It should subtract the result of the following formula to the current RGB values of the colour: $\frac{|value-otherValue|}{2}$.

- +operator*(ratio : double) : Colour
  - This is a method used to create a new colour by multiplying the RGB values of a single colour with a ratio.
  - It is important that if one of the RGB values exceed 255 that it should be set to 255.

- +operator==(rhs : const Colour&) : bool
  - This is a method used to return true if the RGB values of to colours are equal.
  - It should return false if not.

4.2 Mixer

The mixer class simulates a paint palette, that is used by painters to mix colours.

4.2.1 Members

- -palette: Colour[10]
  - This member variable that contains the different colours on the palette.
  - It is an array of colours with a fixed size of 10.
  - Each index of the array correspond to a colour on the palette.

4.2.2 Functions

- +Mixer(red : int, green : int, blue : int)
  - This is the constructor for the Mixer class.
  - It takes three integer parameter, that represent the RGB values of a default colour for the colour palette.
  - It populates the palette array.
  - The colours in the palette at an even index should be initialised to white (All of the RGB values must be 255).
  - The colours in the palette at an odd index should be initialised to a colour with the passed in RGB values.
- +~Mixer()
  - This is the destructor for the Mixer class.
  - There is no dynamic memory to deallocate, therefore it can be left blank.
- +getPalette() : Colour *
  - This is the get method for the palette member variable.
  - It should return the palette array.
- +addColour(index : int, colourToAdd : const Colour&) : bool
  - This method is used to add a colour to the specified index of the palette.
  - If the colour at that index is white, it should be set to the passed in colour (colourToAdd).
  - If the colour it not white, the operator+= of the colour class should be used to mix the colour at the specified index with the passed in colour.
  - Remember to check if the passed in index is valid.
  - If the operation was succesful it should return true and false otherwise.
- +getColour(index : int) : Colour
  - This method is used to get the colour at the specified index of the palette.
  - Remember to check if the passed in index is valid.
  - If it is, the colour at that index should be returned.
  - If it is not, a ”special” invalid colour should be returned with all the RGB values equal to -1.
- +displayPalette() : string
  - This method is used to display each of the colours of the palette.
  - It should return a string of the following format:

```
Colour index: Red: red Green: green Blue: blue
```

  - At the end of each colour, a newline should be added.
  - The index, is the index of the colour in the palette.
  - red, green and blue correspond to the member variables of the Colour class.

- +mixTwo(colour1 : int, colour2 : int) : bool
  - This method is used to mix the two colours at the specified indexes using the operator+ and operator- of the colour class.
  - The colour at the colour1 index should be set to that colour + the colour at the colour2 index.
  - The colour at the colour2 index should be set to that colour - the newly set colour at the colour1 index.
  - Remember to check if the passed in indexes are valid. If not it should return false.
  - If the operation was successful, it should return true.
- +mixRatio(colour : int, ratio : double) : bool
  - This method is used to adjust the colour at the specified index by the specified ratio.
  - It should make use of the operator* of the Colour class.
  - The current colour's RGB values at the colour1 index should be updated to that colour * ratio.
  - Remember to check if the passed in index is valid. If not it should return false.
  - If the operation was successful, it should return true.
- +operator+=(coloursToMix : const Colour[10]) : Mixer&
  - This method is used to mix the colours of the palette with the passed in colours.
  - Each colour of the passed in array should be mixed with the colour at the corresponding index of the colour palette.
  - The += operator should be used to update each colour of the palette.
- +operator+(other : const Mixer) : Mixer&
  - This method is used to combine the colour palettes of two mixers.
  - Each colour at an even index should be set to that colour + the corresponding colour of the other mixer.
  - Each colour at an odd index should be set to that colour - the corresponding colour of the other mixer.
- +operator*=(ratio : double) : Mixer&

- This method is used to adjust all the colours in the palette by the specified ratio.
- It should make use of the operator* of the Colour class.
- The current colour's RGB values should be updated to that colour * ratio.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - string
  - cmath
  - sstream
  - iostream
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- colour.h
- colour.cpp
- mixer.h
- mixer.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 2 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

9 Accessibility

1 Show the class diagram of the two classes to be implemented in this practical:

- Colour
- Mixer

The member variables and functions are discussed in Section 4. The following relationships exists between the Colour and Mixer class:

- Mixer has a relationship with Colour, because of the palette member variable.