UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS110 - Program Design: Introduction
Practical 4

1 Introduction

Deadline: 28th July, 07:00

1.1 Objectives and Outcomes

The objective of this practical is to test your understanding of the programming concepts covered in the theory classes. In particular, this practical will test your understanding of classes and pointers in terms of typical usage.

1.2 Submission

All submissions are to be made to the **assignments.cs.up.ac.za** page under the COS 110 page, and for the correct practical slot. Submit your code to Fitchfork before the closing time. Students are **strongly advised** to submit well before the deadline as **no late submissions will be accepted**.

1.3 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone elses work without consent, copying a friends work (even with consent) and copying textual material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to **http://www.ais.up.ac.za/plagiarism/index.htm** (from the main page of the University of Pretoria site, follow the *Library* quick link, and then click the *Plagiarism* link). If you have questions regarding this, please ask one of the lecturers, to avoid any misunderstanding.

1.4 Implementation Guidelines

Follow the specifications of the practical precisely. For each practical, you will be required to create your own makefile so pay attention to the names of the files you will be asked to create. If the practical requires you to submit additional files of your own, follow the file structure and format exactly. Incorrect submissions will use up your uploads and no extensions will be given.

In terms of C++, unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the practical, will not be allowed.

Some of the appropriate files that you submit will be overwritten during marking to ensure compliance to these requirements.

1.5 Mark Distribution

Activity	**Mark**
Task 1	30
Total	**30**

2 Practical

2.1 Classes

In the theory classes you would have learned about the concept of classes. Essentially classes provide a useful way of organising data and functions within the broader Object Oriented (OO) programming paradigm. In C++, classes are typically structured in the way of a .h and .cpp file pairing, where the .h file defines the class and its functions and member variables and the .cpp file will contain the corresponding implementation. In this, and future practicals, this is the typical structure that you will use in order to implement your practicals. Since this is a C++ specific course, any UML diagrams presented will reflect this and represent pointers, for example, as a specific data type as they would be represented in C++.

2.2 Task 1

You are going to create an implement a simple class called **combiner**. This will consist of **combiner.h** and **combiner.cpp** which will define and implement the class respectively. The purpose of this class is to reconstruct a single, simple greyscale image colour map from multiple fragments. A colour map in this case is a simple grid of integer values ranging from 0 to 255, or black to white. The map is fragmented, and only 3 individual fragments in text file form are available. The objective of the class is to piece the fragments back together into a single colour map. The class has the following features based on this simple UML class diagram:

```
    combiner
-colourMap: int**
-rows: int
-cols: int
----------------------
+setRows(a:int):void
+setCols(a:int):void
```

```
+getRows():int
+getCols():int
+initialiseMap():void
+displayMap():void
+combineFragments(f1:string, f2:string, f3:string):void
+smoothColours():void
```

The variables are as follows:

- rows: the number of rows in the current matrix to construct
- cols: the number of columns in the current matrix to construct
- colourMap: the 2D grid of colour integer values

The methods have the following behaviour:

- setRows, setCols: Two methods that are used set the rows and columns respectively
- getRows, getCols: Two methods used to fetch the values stored for the rows and columns
- initialiseMap: This will create the blank colour map of the rows and columns specified. The initial values in the map should be -1.
- displayMap: This will print out the current map stored in a row column format and with comma delimiters present. For example

  ```
  1,0,255
  17,7,255
  0,2,255
  ```

- combineFragments: This function receives 3 names of text files. In each textfile will be a matrix of values, comma delimited. This is the fragment. Each fragment is sized so that they all combine together into the final size of the actual colour map. The first argument refers to the first fragment and so on. Each must be read into the map sequentially to be correct. The information is read in line by line.
- smoothColours: This function computes the average of the colours in the final colour map and then, rounding to the nearest integer, replaces all of the values in the colour map with the average for the whole map.

You are only allowed to use the **cstdlib, cstring, string, fstream** and **iostream** libraries for this task.

You will need to create a **main.cpp** to test that your class works when used. Submit your files, **main.cpp, makefile, combiner.h, combiner.cpp,frag1.txt,frag2.txt, frag3.txt** as a tar archive to the COS 110 practical submission slot. You will have a maximum of 10 uploads for this task.