UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS110 - Program Design: Introduction
Practical 1

1 Introduction

Deadline: 12th of September, 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 templates in addition to other previously covered topics.

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. If the specification makes use of text files, for providing input information, be sure to include blank text files with the specified names.

1.5 Mark Distribution

Activity	**Mark**
Templates	20
Total	**20**

2 Practical

2.1 Templates

Templates in C++ are the language's way of enabling the use of generic programming to greatly enhance its own capacities and flexibility. A good example of this the vector class in C++. Vectors can be created and instantiated with a wide variety of types from ints to strings and yet the underlying code and use of a vector does not change from its instantiated type. Templates are the key to achieving this. With templates, functions and classes are able to provide a generic, type-independent, interface that can defer a specific instantiation of a type until later. This practical is going to make use of templates to demonstrate this flexibility.

Additionally, you will be not be provided with mains. You must create your own main to test that your code works and include it in the submission which will be overwritten during marking.

2.2 Implementation Note

There are a number of ways to implement templates. In this practical, a specific way is tested. As always, class definitions are put into .h files and implementation done in the .cpp file. Remember that in the .cpp file, the methods you implement will require the usage of the specifier: "**template < class T>**" in order to use template structures. Finally, when testing your implementation, remember to include the .cpp file as well as the .h file in the main where you test.

2.3 Task 1

You are going to implement a class **richMatrix** which is going to demonstrate the use of templates. This class is going to implement a number of methods that all make use of templates in order to make the class very flexible. In this case, the usage of $< T >$ indicates a template class with the specific usage of T as the indicator for the template type.

2.3.1 richMatrix

The class is described according to the simple UML diagram below:

```
richMatrix <T>
------------------------
+createNewMatrix(size:int): T **
+addMatrix(mat1: T**,mat2:T**,size:int): T**
+subtractMatrix(mat1:T**,mat2:T**, size:int):T**
+transposeMatrix(mat1:T**, size:int):T**
+print(mat:T**,size:int):void
```

The class methods do the following:

- createNewMatrix: This receives a size and makes a new square matrix of type T that is returned. The matrix is not populated with anything at this point.
- addMatrix: This function receives two square matrices, with their size defined by the the passed in size argument. This method will do an element-wise addition between both matrices and return a third matrix containing the results of this operation. In the case of strings, this will concatenate the two elements together.
- transposeMatrix: This function receives a matrix, and a size, and then produces a transpose of the given matrix. Importantly, the matrix returned is the original one with the transposition performed, and not a new matrix. For more detail on the transpose operation, please see this link

  `https://chortle.ccsu.edu/VectorLessons/vmch13/vmch13_14.html`

- subMatrix: This function receives two square matrices, with their size defined by the the passed in size argument. This method will do an element-wise subtraction, mat1-mat2, between both matrices and return a third matrix containing the results of this operation.
- print: This method will receive a square matrix whose size is an argument passed in. It will then print this matrix out line by line, with all the elements of one row on one line, with no delimiters.

You will be allowed to use the following libraries: **cstdlib,string,iostream**. You will have a maximum of 10 uploads for this task. Your submission must contain **richMatrix.h, richMatrix.cpp, main.cpp** and a makefile.