UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

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

1 Introduction

Deadline: 11th of September, 18:00

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 operator overloading 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**. Only submit either .tar or .zip files to the marker.

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**
Tome	20
Total	**20**

2 Practical

2.1 Operator Overloading

An operator is the term used to describe a symbol in a programming language that is representative of some action. Most users will be familiar with the usage of operators as they are commonly employed to improve the syntax and usability of a language. For example, in C++, the << operator is commonly employed when printing information to the screen.

In C++, operator overloading is the practice of overloading the usage of an operator so that it can, in a limited context, provide new functionality in ways not described for the original usage. This can greatly extend the write-ability of a large programming task since complex operators can be performed through overloading of the appropriate operators.

2.2 Task 1

In this task, imagine that you are an apprentice to a digital wizard. As part of your apprenticeship, you will be asked to help maintain a digital library of spells. Unfortunately, most of the spell books, called tomes, are fragmented and incomplete; your master is most forgetful in her old age. Now you are tasked to implementing a system that will help improve manipulation of the library. Your master is however very forgetful and so instructs you to implement a simpler interface that can do a number of tome management tasks, doing so via the use of operator overloading.

2.2.1 Tome Class

Described below is a simple UML diagram for the **tome** class. This is the class that will be used to overload a number of operators so that managing the archive will be easier.

```
tome
-references: string[5]
-tomeName: string
-author: string
```

```
-currSpells:int
----------------------------
+tome(name:string, author:string)
+tome(name:string, tomeSize:int,author:string, initialList:string *)
+~tome()
+getTomeSize() const:int
+getSpell(i:int) const:string
+getName() const:string
+getAuthor() const:string
+friend operator<<(output:ostream &,t:const tome &):ostream &
+operator+(add:const string &):tome
+operator-(sub:const string &):tome
+operator=(oldTome:const tome &):tome&
+operator>(t:const tome&):bool
+operator<(t:const tome&):bool
+operator==(t:const tome&):bool
```

The class variables are described as follows:

- references: A array of string names that represent the names of spells that reside in the tome. Each is a separate spell. The default state of the array is to be populated by empty strings.
- tomeName: The name of the tome as a string. Examples include "The Magisterial Cookbook" and "Eye of Newt & Other Curses"
- author: The name of the original author(s) of the tome as represented by a string.
- currSpells: The current number of spells in the tome. It can be at most the maximum of 5 but never larger than it.

The class methods and their behaviour is described as follows:

- tome(name:string,author:string): This constructor receives a number of arguments and instantiates them in the class.
- tome(name:string, tomeSize:int,author:string, initialList:string *): The class constructor. It will receive a number of arguments and then instantiate all of the class variables. The tomeSize variable is the number of spells in the initialList variable.
- ∼tome: The default class destructor.
- getTomeSize: Returns the number of spells stored in the tome.
- getSpell: This accepts an array index and returns the spell stored at the index.
- getName: This returns the name of the tome.
- getAuthor: Returns the name of the author(s) of the tome.

- operator+: This operator will enable a tome to add a spell that is given as a string into itself. If the tome is full, nothing should happen, but if the tome is not full, then the new spell should be added at the first empty space.

- operator-: This operator will enable to a tome to remove a spell that is provided as a string using the - operator. The first instance, from index 0, of the spell should be removed from the given tome. If the spell is not found, then nothing should happen to the tome.

- operator<: This compares two tomes. One tome is lesser than the other if one tome has fewer spells than the other. It returns false if they are equal.

- operator>: This compares two tomes. One tome is greater than the other if one tome has more spells than the other. It returns false if they are equal.

- operator<<: This is an overload of the outstream operator. When called on a tome, such as cout<<A, where A is a tome, the resulting operation must send to cout, all of the information of the tome in a neatly formatted way. The format is:

  ```
  Tome Name: Hexes and Incantations, A Study
  Author: S. Swaddles, D. Pines
  References: Hex Curse, Summon Ice Golem
  ```

- operator=: This is an overload of the assignment operator. With this operation overload, it should be possible to assign an existing instance of a tome to a new instance. All of the variables of the old tome should be instantiated in the current tome.

- operator==: This is an overload of the equality operator. It should return true if two tomes are equal and false otherwise. Two tomes are only equal if they hold the same number of spells and have the same number of spells in the same exact quantities. For example, consider two tomes with spells A,A,C and A,C,A. These two tomes are equal because they have 3 spells stored each, and each spell appears in the same quantity as the other spell, regardless of its position in the array.

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