UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS110 - Program Design: Introduction
Practical 3

1 Introduction

Deadline: 28st August, 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.

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**
Task 1	35
Total	**35**

2 Practical

2.1 Classes - Copy Constructors and Operator Equals

For many classes the instantiation of the class will occur using a constructor that receives values as arguments that correspond to the needed variables of the class. There are times when an object needs to be instantiated using another object. Constructing one object with another is often faster than constructing it from scratch. Doing so involves the use of a specialised constructor called a copy constructor.

However, this is not the only way to use one object to instantiate another. The = operator, which is used in arithmetic can also be used to instantiate objects. Therefore this practical also introduces the concept of operator overloading which is where an operator defined by C++ has its implementation changed in a class to do something other than the default implementation.

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 Task 1

In this task you are going to implement a class called **cauldron**. This consists of a **cauldron.cpp** and **cauldron.h** and **ingredient.cpp** and **ingredient.h**. Each of these classes is defined in a simple UML diagram below:

2.2.1 Cauldron Class

```
cauldron
-ingredients: ingredient **
-numIngredients: int
-maxIngredients: int
-----------------------------
+cauldron(ingredientList:string,maxIngredients:int)
+cauldron(oldCauldron: const cauldron*)
```

```
+operator=(oldCauldron: const cauldron&):void
+~cauldron()
+print():void
+getMax() const:int
+getCurr() const:int
+getIngredient(a:int) const:ingredient *
+addIngredient(in:string,dR:int):int
+removeIngredient(in:int):void
+distillPotion(currCauldron: cauldron &, list: string *,numRemove:int):void
+listIngredients():void
```

The variables are as follows:

- ingredients: A dynamic array of the ingredients in the cauldron.
- numIngredients: The current number of ingredients in the cauldron.
- maxIngredients: The maximum number of ingredients in the cauldron.

The methods have the following behaviour:

- cauldron(ingredientList:string, maxIngredients:int): The basic constructor for the class. It receives a string, which is a name (including the file extension) for a file that will contain the input values. The maximum number of ingredients in the cauldron is specified as well. The file will consist of an unknown number of lines. On each line, there will be an ingredient in the following format (for example):

  ```
  giant toe,0
  ```

  The first component of the string is the ingredient name followed by an integer value representing the danger rating of the ingredient. This is comma delimited. If there are more ingredients in the file than the maximum allows, only store in the first X, where X refers to the maximum. For example, if there's a limit of 10, and the list has 30, only store the first 10 ingredients.

- cauldron(oldCauldron:const cauldron *): A copy constructor for the class. It will copy the variables of the argument into the newly created instance.
- ~cauldron: The destructor for the class. It deallocates all of the memory assigned by the class. The ingredients array should be deallocated from index 0.
- operator=(oldCauldron:const cauldron&): Than overload of the assignment operator. When used in the format $A = B$, the variables of B must be assigned to A after the line executes.
- print(): This prints out a summary of information with the following format (with example values):

  ```
  Number of Ingredients: 10
  Average Danger Rating: 2.42
  Maximum Danger Rating: 5
  Minimum Danger Rating: 0
  ```

- getMax(): Getter for the maxIngredient variable. It is a constant function.
- getCurr(): Getter for the numIngredient variable. It is a constant function.
- getIngredient(a:int): Gets and returns the ingredient pointer at the index a. It is a constant function.
- addIngredient(in:string,dR:int): This adds the given arguments as a new ingredient to the ingredients array. The new ingredient should be added to the first open space in the array. This function returns the index where the new ingredient is added. If there is no space in the current array, the array should be extended by 1 and then the new element should be added at the end. The maximum size of the array should increase as well.
- removeIngredient(in:int): This function deletes the ingredient associated with the given index. If the index is out of bounds, it should return. The number of ingredients should change as well.
- distillPotion(currCauldron: cauldron&, list: string *,numRemove:int): This function receives a reference to a cauldron object and a list of ingredient names (with the number to remove as well). Each ingredient in the ingredients argument should be deleted from the ingredients list of currCauldron if found. If there are duplicates, the first one should always be deleted.
- listIngredients(): This function prints out the current ingredients in the cauldron, from index 0. Each name is printed out on one line with a new line at the end. An example is given below:

```
blueberry
blackberry
strawberry
cherry
```

2.2.2 Ingredient Class

```
ingredient
-name:string
-dangerRating:int
------------------------------
+ingredient(name:string,dangerRating:int)
+ingredient(newIngredient:const ingredient *)
+ingredient(newIngredient:const ingredient &)
+~ingredient()
+getDanger():int
+getName(): string
```

The variables are as follows:

- name: The name of the ingredient.

- dangerRating: The rating of the ingredient in terms of how dangerous it is. 0 is a perfectly safe ingredient and 10 is the most dangerous.

The methods have the following behaviour:

- ingredient(name:string,dangerRating:int): The value constructor of the class. It constructs the class using the passed in variables.
- ingredient(newIngredient:const ingredient *): A copy constructor for the class. Create the new instance of the class using the passed in argument.
- ingredient(newIngredient:const ingredient &): A copy constructor for the class. Create the new instance of the class using the passed in argument.
- ∼ingredient: The default destructor for the class.
- getDanger(): getter for the dangerRating variable.
- getName(): getter for the name variable.

You are allowed to use the following libraries for each class:

- ingredient: string
- cauldron: iostream, fstream, sstream

You are allowed to use namespace standard in your header files. You will have a maximum of 10 uploads for this task. Your submission must contain **ingredient.cpp, ingredient.h, cauldron.h, cauldron.cpp, list.txt,main.cpp** and a makefile.