COS 214 Practical Assignment 4

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- Date Issued: **30 August 2022**
- Date Due: **13 September 2022** at **11:00am**
- Submission Procedure: **Upload via ClickUP**
- Submission Format: **archive (zip or tar.gz)**

1 Introduction

1.1 Objectives

In this practical you will get to understand the following:

- the integration of the Memento, Composite, Observer and Iterator Design patterns.
- UML Sequence and Activity diagrams.
- the programming tools: GDB for debugging your code, Valgrind for tracing memory leaks in your code, and GoogleTest to perform unit testing of your code.
- perform unit testing on your code and report both positive and negative test results.
- plan and design a real-world application using UML and learnt design patterns.
- implement designed solution in C++ translating UML to code.

1.2 Outcomes

When you have completed this practical you should:

- have a basic practical knowledge of how to use GDB, Valgrind, and GoogleTest.
- be able to use the Memento Design pattern to solve practical problems, while understanding how to restore the state of an object to a previous state.
- be able to use the Composite Design pattern to solve practical problems, while to treating multiple separate objects as one whole by way of using tree structures.
- be able to use the Observer Design pattern to solve practical problems, while reducing coupling between classes and improving code reusability.
- be able to use the Iterator pattern to access a collection of objects indirectly through a mediated interface.
- be able to design test cases and develop a test suite for parts of your code.
- be able to model real-world problems using appropriate UML structures.

2 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be allowed to give you the solutions.

3 Submission Instructions

You are required to upload all your source files (that is `.h` and `.cpp`), your Makefile, UML diagrams as individual or a single PDF document and any data files you may have created, in a single archive to ClickUP before the

deadline. **If you did not upload you will receive a 0 for this practical assignment. Code that does not compile and run will not be marked.**

4 Mark Allocation

Task	**Marks**
Programming Tools	15
File System	20
File System API	20
Snapshot File System	20
Auxiliary Functions	20
Demo	30
Unit Testing	10
TOTAL	135

5 Assignment Instructions

In this assignment you will learn how to use the GNU Debugger to debug your code in case it produces errors that are not obvious from looking at the source code itself. You will then use Valgrind to trace any memory leaks in your *C++* program.
The remainder of the assignment will guide you through the implementation of the Composite, Observer, Iterator, and Memento Design patterns.
Finally, you will develop a unit test suite for parts of your program.

Note: During marking you may be asked to import all your code into Visual Paradigm. Implementing this practical using a procedural approach and not using design patterns will result you in forfeiting your mark for this practical.

Double Note: It is advised that you plan your time accordingly and work continuously on this assignment. Failure to work consistently could result in you running out of time.

Task 1: Programming Tools ..... (15 marks)

A professor at 214 University is worried about the performance of his students in the module "Design Patterns in C++". He decides to develop a C++ program that will improve the students' performance by calculating each of their marks as a percentage of the highest mark per test. In one of the tests, the performance was so poor that no student obtained any marks. He was very eager to see how far his program would go in helping the students. To his amazement, his program also stopped working.

1.1 You are provided with the program `marks.cpp` which the professor was using to improve the students' marks. Follow the instructions below to find out what the cause of the error is that is tormenting the old professor: (10)

1. Compile and run the program on your Linux machine with an option to request debugging information; note the behaviour of the program.
2. Use GDB to run the program in debug mode.
3. Use a command to run the program within the debugger. What is the error produced by the program? At which line did the error occur and what were the values of the function arguments at the time?
4. Use the list command to see the context of the crash, i.e., to list the lines around where the crash occured.
5. Use the `where` or `backtrace` command to generate a stack trace and interpret the output.
6. Move from the default level '0' of the stack trace up one level to level 1.
7. Run the `list` command again and note which lines are printed this time.
8. Print the value of the local (to main) variable *highest*
9. Why did the crash occur?

Write all the answers to the questions in a PDF document that you will include in your upload.

1.2 While waiting for you to debug his "marks" program, the professor decides to write another program that he will use to input marks for all his 10 students. This program also behaves unexpectedly and the professor gives it to you with a suspicion that there is a memory leak. (5)

1. Compile the program `capture.cpp` with an option to include debugging information.
2. Run your program in Valgrind with the `leak check` option enabled. There is no need to specify "Memcheck" for the tool, as it is the default option.
3. What is the process ID on your output?
4. What is the first error that you see from the output?
5. Below the first error is a stack trace, what does it tell you?
6. "definitely lost" shows that there is a memory leak in your program. Why has 40 bytes been lost?
7. How would you fix the memory leak?

Write all the answers to the questions in a PDF document that you will include in your upload.

Task 2: File System ..... (20 marks)

File system structures are at the heart of modern computing. These are required to be present from the smallest computing devices, such as Android Media Players, Car Entertainment systems to the largest of systems, like the International Space Station or the largest cloud computing provider's storage mechanisms. In this practical you will design and implement a file system using design patterns. This will allow you to expose an easy API for yourself, to enhance your file system to handle various complicated tasks easily.

Most modern file system use tree structure designs. The reason, is that tree structures have been shown to be very efficient in the domain of file systems. Your file system will allow for two modes of accessing files, namely synchronous and asynchronous access. Your file system will consider access files on any local device, such as ahard disk drive (HDD) or solid state drive (SSD) as synchronous access. Your file system will handle remote files as a first class citizen using technologies like CIFS, NFS and Google Drive. The hierarchy of your file system will rely on the root concept of a `Node` object. Your file system needs to allow for two concepts, namely that of a directory and a file. A directory is to be considered a holder, which can contain other directories or other files. A constraint is that a synchronous directory can only contain an asynchronous directory while an asynchronous node(s) can not contain any synchronous node(s).

In this task draw upon your knowledge of your CS degree, including data structure and operating systems, and design your file system using design patterns. Various questions follow below that must be answered, either with code or in your single submitted PDF.

You are not required to write out the contents of the files to any persistence medium. However, your files should contain at least a name and some string content.

2.1 The above concepts can be modelled using a variety of combinations of design patterns where two or more patterns are used together. Discuss and contrast two different combinations of design patterns that can be used to model the above scenario.

2.2 After careful consideration of the advantages and disadvantages of your two approaches, choose one combination and model it using Visual Paradigm.

2.3 Implement your design **after** modelling it in Visual Paradigm.
Note: UML diagrams that are drawn after you have coded will be penalised.

Task 3: File System API ..... (20 marks)

Having a file system that nobody can extend is not very useful in the modern computer science industry where third-parties plugin and extend an application. In this task you would like that yourself and other developers be able to plugin to your design of the file system. In any software project, before other developers can expand on a system, you need to provide them with a useful Application Programming Interface (API).

3.1 Expand your UML model to allow for handling directories as aggregates in preparation for implementing the Iterator design pattern. In addition your UML should be amended to add the following API to your directories. The iterator design pattern and below API's will be used in the subsequent tasks for the implementation of auxiliary functionality.

- `addDirectory(Directory): void`: adds a new directory to the collection
- `removeDirectory(): void`: removes an directory from the collection.
- `listDirectories(): bool`: checks whether the collection is empty.
- `isEmpty(): bool`: checks whether the collection is empty.
- `addFile(File): void`: adds a new directory to the collection
- `removeFile(): void`: removes an engineer from the collection.
- `listFiles(): bool`: checks whether the collection is empty.

Remember that these functions are defined as virtual.

Note: UML diagrams that are drawn after you have coded will be penalised.

3.2 Create a `NodeIterator` with the following specification.

- a function `first()` which specifies the first node in the aggregate.
- a function `next()` which specifies the next node in the aggregate.
- `hasNext()`: checks whether the aggregate has a next node after the node directory.
- `current()`: returns the current node.

All these functions are virtual.

3.3 Adapt your UML and explain which creational design pattern will work best to allow the user to create two different iterators, namely a `DirectoryIterator` and a `FileIterator`. The `DirectoryIterator` and `FileIterator` are specialisations of the `NodeIterator`. As per the naming convention, the `DirectoryIterator` will function only on directories. The `FileIterator` will function in a similar fashion on files.
Note: UML diagrams that are drawn after you have coded will be penalised.

3.4 In your implementation, provide two mechanism how one can go about to return directories for the `DirectoryIterator` while returning files for the `FileIterator`. Discuss the advantages and disadvantages of each approach.

3.5 Implement the above API into your File System with the appropriate creation mechanism of the iterators.

Task 4: Snapshot File System ..... (20 marks)

Modern file systems automatically takes snapshots as files are modified. In this task your are required to implement a snapshotting mechanism which can be triggered by the user to create a complete snapshot of your filesystem.

4.1 Create an aggregate class called `Root` which functions as the root of the entire file system. This aggregate should provide users the ability to:

- create a complete snapshot of the entire file system
- restore a complete snapshot of the entire file system
- clear all previous snapshots of the entire file system

4.2 You must design your snapshotting mechanism with the help of a UML Activity diagram.

4.3 Modify your UML class diagram to accommodate for the above requirement whereafter you should implement your design.
Note: UML diagrams that are drawn after you have coded will be penalised.

Task 5: Auxiliary Functions ..... (20 marks)

Modern file systems have various functions that it provides to the user, such as recommendations, indexing, access controlled lists (ACLs) and other mechanisms. One approach is to have the operating system continuously scan the entire file system for changes. With modern operating systems having millions of files, this approach is inefficient. Use design patterns to rather have the file system alert interested services that a file or directory has been changed. Changes include files or directories being created, deleted, or modified.

5.1 Expand your system by allowing external observers to be attach themselves to a `Node` class. If any changes occur to either a directory or file, notify the observers attached to the current directory. Thereafter traverse upwards through the hierarchy and alert observers attached to each directory until you reach the root.

5.2 Create an anti-virus application that can attach itself to the root of the filesystem. Every time a change occurs with files, that is a file is created or modified, the anti-virus should check the file for malware. You are free to implement a mechanism to simulate this in your main application. The anti-virus should alert a user whenever a file or directory is removed.

5.3 Show by way of an UML Sequence diagram how your systems' notification mechanism works to alert all interested parties of changes to the underlying file system.

Task 6: Demo ..... (30 marks)

6.1 Your are required to construct a *driver program* to demonstrate your file system. Creativity and effort will be awarded extra in the practical.

6.2 You are required to demonstrate by way of using GDB that your file system functions correctly. Areas to consider would be the removal of files, the creation and clearing of snapshots. During marking you may be requested to demonstrate using GDB. However be sure to include screenshots in your PDF explaining the process, the commands and the line numbers at which screenshots were taken at.

Task 7: Unit Testing ..... (10 marks)

Pick any function that you have implemented and develop a unit test suite using GoogleTest. You may perform a boundary value analysis, branch testing, or loop testing depending on the function you have chosen. Submit both positive and negative test results along with your unit test code.