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

1 Introduction

Deadline: 22nd October, 19:30

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 polymorphism, function templates and exceptions in addition to other previously covered topics.

1.2 Submission

All submissions are to be made to the **ff.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 else's work without consent, copying a friend's 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

Task	**Mark**
Task 1	64
Total	**64**

2 Practical

2.1 Templates

Templates are a feature in C++ that enable a function or a class to accept generic parameters. Rather than coding a function to accept a specific kind of input parameter type, it can be coded to accept a generic type that can then be implemented to respond to different kinds of inputs. This can be extended for classes which enable them to be parameterised with different types.

2.2 Exceptions

Exceptions in C++ are programmatic responses to circumstances that arise during program execution that are problematic in some way. Problematic in the sense that an error has generally occurred. Exceptions and exception handling provide a way for programmers to handle errors during runtime without having to halt program execution.

Exceptions are incredibly useful for coding projects because they can be used to help determine errors which might occur in program runtime. There are a number of different types of exceptions provided by C++ but creating custom exception classes is also allowed.

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.

3 Task 1

For this task, you are going to implement safety procedures for transport systems that are meant to indicate when they have violated some safety condition related to their use. There are two primary classes, the trolley and the train. The main concern for the trolley is making sure the speed is not too fast or too slow. The train meanwhile can carry both fluid and solid cargo and thus needs to be guarded for running out of cargo to offload or not enough space to maintain its cargo. A third class, the manager, is used as part of a testing procedure for multiple trains at a time.

3.1 Train Class

The train class is given by the following UML diagram. The train class is a template class. It is primarily going to be used by numerical types of int and double.

```
class train<T>
-currStorage:T
-storageCap:T
-name:string
-------------------------------------------
+train():
+~train():
+train(name:string,storageCap:T):
+getName():string
+getCargoCap():T
+getCurrCargo():T
+loadCargo(cargo:T):void
+unloadCargo(cargo:T):void
```

The class variables are as follows:

- currStorage: The current amount of storage contained within the train.
- storageCap: The maximum amount of storage the train can contain.
- name: The name of the train.

The class methods are as follows:

- train(): A default constructor. It does not do anything.
- ∼train(): The default destructor. When called, the destructor should print the following message:

  ```
  Dispatch Name: X
  Current Storage: Y
  Storage Max: Z
  ```

  where X,Y and Z refer to the train's name, currStorage and storageCap variables. Each is printed one its own line with a new line at the end.
- train(name:string,storageCap:T): A constructor that sets the name of the train and the storage capacity. The current storage should be set to 0.
- getName(): Returns the name of the train.

- loadCargo(cargo:T): This will attempt to load the new cargo into the train. If this would exceed the maximum amount allowed, then the storageFull exception should be thrown. Otherwise, the current storage amount should be increased by the provided argument.

  When the function is called, it should also print out a message of the current amount of storage in the train if no exception is thrown. The message format is as follows:

  ```
  Capacity: X
  ```

  where X is the current amount of storage taken up.

- unloadCargo(cargo:T): This will attempt to unload an amount cargo from the train. If the amount requested, via the argument, is greater than what the train has, then the storageEmpty exception should be thrown. Otherwise, the current storage amount should be decreased by the provided argument.

  When the function is called, it should also print out a message of the current amount of storage in the train if no exception is thrown. The message format is as follows:

  ```
  Capacity: X
  ```

  where X is the current amount of storage taken up.

- getCargoCap():Returns the cargo capacity.
- getCurrCargo(): Returns the current amount of cargo.

3.2 Trolley Class

The trolley class is given by the following UML diagram.

```
class trolley
-speed:int
-name:string
------------------------------------
+trolley():
+~trolley():
+trolley(name:string):
+getSpeed():int
+increaseSpeed(s:int):void
```

The class variables are as follows:

- speed: The speed value of the trolley.
- name: The name of the trolley.

The class methods are as follows:

- trolley(): The default constructor. It does not do anything.

- ∼trolley(): The default destructor.
- trolley(name:string): This constructor sets the name of the trolley and also initialises the speed to 0.
- getSpeed(): This returns the speed of the trolley.
- increaseSpeed(s:int): This increases the speed of the trolley by the provided amount.

3.3 Manager Class

The manager class is given by the following UML diagram.

```
class manager<T>
-trains:train<T> **
-numTrains:int
------------------------------------
+manager(input:string):
+~manager():
+summarise():void
+loadTrain(i:int,load:T):void
```

The class variables are as follows:

- trains: A dynamic array that stores a number of trains of type T.
- numTrains: The number of trains to be stored.

The class methods are as follows:

- manager(input:string): This constructor will take the name of a file as a string. When called this file should be used to initialise the class. The first line of the file will be the number of trains. Every line after that will consist of a two elements, the name of the train and the cargo maximum for that train. An example of this is

  ```
  001,24
  002,25
  ```

- manager(): This will deallocate all of the memory of the class.
- summarise(): The summarise function calculates the total of the cargo capacity of all of the trains held in the manager class. It prints this out with a message as follows:

  ```
  Total Current Storage: X
  ```

  where X is the total amount of storage capacity of all of the trains.
- loadTrain(i:int,load:T): This function will receive an index, i, and a load value. It will add that load to the train i.

3.4 Exception Classes

There are four exception classes that you have to implement. Each of these derives publicly from the exception class. No UML diagram will be provided for the exception class but a description of the required behaviour will be.

As an additional hint, there should be an overload for the what() function that is used by the exception classes.

3.4.1 tooSlow

The tooSlow exception is going to be used in conjunction with the trolley class. When called, the exception should provide a message in the what function that says "TOO SLOW".

3.4.2 tooFast

The tooFast exception is going to be used in conjunction with the trolley class. When called, the exception should provide a message in the what function that says "SPEED LIMIT EXCEEDED".

3.4.3 storageEmpty

The storageEmpty exception is going to be used by the train class. When called, the exception should provide a message in the what function that says "STORAGE EMPTY".

3.4.4 storageFull

The storageFull exception is going to be used by the train class. When called, the exception should provide a message in the what function that says "STORAGE FULL".

4 Submission Guidelines

The **train** class should make use of string and iostream libraries as includes.

The **trolley** class should make use of the string and iostream libraries as includes.

The **manager** class should make use o the string, sstream and fstream libraries.

Each exception class should include exception and the iostream library.

Your submission must contain:

- storageEmpty.h
- storageFull.h
- tooSlow.h
- tooFast.h

- train.h
- train.cpp
- trolley.h
- trolley.cpp
- manager.h
- manager.cpp
- makefile
- input.txt
- main.cpp

You will have a maximum of 10 uploads for this task.