Department of Computer Science

COS110 2016
Practical 4

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1 Introduction

1.1 Submission

The submission deadline for this practical is **13 September 2016 at midnight**. You should aim to complete this assignment before the due date.

Fitchfork marks by comparing the output of your program with specified expected output on a line by line basis. For this reason you should pay close attention to the instructions for the output of your program. Also remember that names of files are case sensitive.

1.2 A Serious Warning

It is in your own interest that you, at all times, act responsible and ethically. As with any work done for the purpose of your university degree, remember that the University of Pretoria will not tolerate plagiarism. Do not copy a friend's assignment or allow a friend to copy yours. Doing so constitutes plagiarism, and apart from not gaining the experience intended, you may face disciplinary action as a result. For more information read the University of Pretoria's plagiarism policy on url http://www.library.up.ac.za/plagiarism/index.htm

1.3 Uploads

The student is advised to ensure that tarballs are created properly before uploading. The *.tar.gz* created by a student should not have any sub-folders. The zip folder should **ONLY** contain the source code.

Given code and data

Extract the content of the `Prac4.tar.gz` archive. After extracting this archive in a directory named `COS110/Prac4`, this directory should contain the files and directories shown in the following hierarchical structure.

```
COS110
└── Prac4
    ├── Prac4.tar.gz
    ├── Task2
    │   ├── makefile
    │   ├── global_includes.h
    │   ├── main-prac4-task2.cpp
    │   ├── person.h
    │   ├── staff.h
    │   ├── student.h
    │   └── Reedme.txt
    └── Task3
        ├── global_includes.h
        ├── learning_machine.h
        ├── lecturer_msc.h
        ├── lecturer_phd.h
        ├── main.cpp
        ├── person.h
        ├── staff.h
        └── student.h
```

Task 1: Filters [10]

This task will create an abstract filter class with two inherited classes: titleCase and backwards.

You have received no additional files therefore you have to create all the necessary files.

The following files should be created:

- **filter.h**
- **titleCase.h**
- **titleCase.cpp**
- **backwards.h**
- **backwards.cpp**

The students are advised to make their own **main.cpp** and **makefile** to test the functionality of their code.

Descriptions of the files are given below:

- `filter.h`: this file should contain a class called *filter* which must declare one pure virtual function. The following should be a pure vitual function –> `string print(string s)`.
- `titleCase.h`: should contain the **`class`** `titleCase` and should define the print function.

- `titleCase.cpp`: implementation for converting a `string to title` case:

  1. All words must start with a capital.
  2. Capitals in the middle of words must be converted to lower case.
     Example:

     ```
     hELlo -> Hello
     ```

  3. Special words should start with lower case letters. For this practical only consider the special words *"and"*, *"the"* and *'a'*.

- `backwards.h`: should contain a class called backwards and should define the function print.

- `backwards.cpp`: implementation for converting string into backwards format. Each individual words should be reversed.

  ```
  hello there friend -> olleh ereht dneirf
  ```

  Note: Each individual words should be converted AND NOT THE WHOLE SENTENCE. There should be no endlines in your strings. There must not be any cout in the code other than in the main.

On completion, create a tarball containing the **filter.h**, **titleCase.h**, **titleCase.cpp**, **backwards.h** and **backwards.cpp**. Upload it using the active fitchfork assignment called **Practical 4 - Filter**.

Task 2: Inheritance [30]

This task will model the interaction between people, i.e. how they greet each other, and the activities of a person at work.

You have been given three header files containing the definition of three corresponding classes. You need to implement all the required methods in the corresponding **.cpp** files.

The classes follow.

- `Class Person`: This is an *Abstract class* that will act as the parent of the other two classes in **task 3**. For detailed descriptions of Person's members and methods, view the file **person.h**. Some of the pure virtual functions should have implementations (this will be clearly stated).

  Class Person must have the ability to remember the persons previously greeted. This is accomplished by storing the pointer of the greeted person in a dynamic array of `type Person**`. In the **h-file**, these persons are referred to as colleagues.

  Note: You are advised to read the file **person.h** *from top to bottom before attempting this task. Implement all methods in* **person.cpp**.

- `Class Student`: This is a *concrete class* that inherits from Person. All the methods that must be implemented is contained within the file **student.h**. Detailed descriptions of every method inside **student.h** will clarify what is expected of you. Implement all methods in **student.cpp**.

- `Class Staff`: This is a *concrete class* that inherits from Person. All the methods that must be implemented is contained within the file **staff.h**. Detailed descriptions of every method inside **staff.h** will clarify what is expected of you. Implement all methods in **staff.cpp**.

You are given a **main** file that extensively produces output so that you may verify that everything is working as expected. You can deduce the overall working of the three classes by studying this output. You will find the expected output at the bottom of the **main** file as one large comment.

Note: All methods that returns strings should not have new line characters in them, even if they are very long. This is to minimize the chance for our output to be misaligned when uploading to FitchFork.

On completion, create a tarball containing the **person.cpp**, **student.cpp**, and **staff.cpp**. Upload it using the active fitchfork assignment called **Practical 4 - Inheritance**.

Task 3: LearningMachine [15]

This task extends the classes from **Task 2** to show the possibilities of *polymorphism*, as well as demonstrate *dual inheritance*. The following header files are provided:

- `learning_machine.h`
- `lecturer_phd.h`
- `lecturer_msc.h`
- `student.h`
- `staff.h`
- `person.h`

Do not edit these header files. *It is advised that you read through these header files before doing the practical, especially the altered ones from Task 2.*

The following changes are to be made to the files and classes from Task 2:

- A *new class* **LearningMachine** is added. LearningMachine is an *abstract class* that provides an interface to allow people to learn and teach each others, modifying their proficiency. Refer to the given **header file** for details.
- Classes *Student* and *Staff* now inherit from both *Person* and *LearningMachine*, allowing them to both *learn* and *teach*.

- A new class **LecturerPhD** is added. LecturerPhD *inherits* from *Staff*, and contains the following:

    1. `LecturerPhD(string name, Person::Gender gender)`:
       The constructor should initialize the LecturerPhD as a staff member with the corresponding name and gender, and a proficiency of 80.
    2. `string teach(Person* student)`:
       Overrides the function from LearningMachine. Refer to the given header file for details.
    3. `string learn(Person* student)`:
       Overrides the function from LearningMachine. Refer to the given header file for details.
    4. `double getSomeProficiency()`:
       Returns 10% of the lecturers proficiency.
    5. `string getTitle()`
       Returns "Dr".

- A new class **LecturerMSc** is added. LecturerMSc *inherits* from *Staff*, and contains the following:

    1. `LecturerMSc(string name, Person::Gender gender)`:
       The constructor should initialize the LecturerMSc as a staff member with the corresponding name and gender, and a proficiency of 70.
    2. `string teach(Person* student)`:
       Overrides the function from LearningMachine. Refer to the given header file for details.
    3. `string learn(Person* student)`:
       Overrides the function from LearningMachine. Refer to the given header file for details.
    4. `double getSomeProficiency()`:
       Returns 5% of the lecturers proficiency.

- The Staff class is modified as follows, to comply with LearningMachine:

    1. Staff becomes an *abstract class*, as it does not implement the pure virtual function from LearningMachine.
    2. The constructor takes an extra parameter used to initialize LearningMachine. `Staff (string name, Gender gender, double proficiency)`.
    3. Staff now *inherits* from LearningMachine as well as Person.

- The Student class is modified as follows, to comply with LearningMachine:

    1. The constructor retains the same arguments, but additionally initializes LearningMachine with a proficiency of 0.
    2. `string teach(Person* student)`:
       Overrides the function from LearningMachine. Refer to the given header file for details.

3. `string learn(Person* student)`:
   Overrides the function from LearningMachine. Refer to the given header file for details.
4. `double getSomeProficiency()`:
   Refer to the given header file for details.

A **main** file is provided to test your implementations. This will not test all corner and edge cases, so be sure to test your code thoroughly. There should be no output statements in your functions, and no newline characters should be returned in any strings.

On completion, create a tarball containing the **learning_machine.cpp**, **lecturer_msc.cpp**, **lecturer_phd.cpp**, **person.cpp**, **student.cpp**, and **staff.cpp**. Upload it using the active fitch-fork assignment called **Practical 4 - LearningMachine**.