Department of Computer Science

COS110 2016
Practical 8

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1 Introduction

1.1 Submission

The submission deadline for this practical is **07 November 2016 at 11:59pm**. 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 `Prac7.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
└── Prac8
    ├── Prac8.tar.gz
    ├── Task1
    │   ├── makefile
    │   ├── LinkedList.cpp
    │   ├── LinkedList.h
    │   └── main.cpp
    └── Task2
        ├── makefile
        └── main.cpp
```

Task 1: LinkedList [10]

For this Practical, you need to implement the combine function in **LinkedList.cpp**.

Note: YOU CAN ONLY CHANGE CODE IN **LinkedList.cpp**, *NO WHERE ELSE!*

You have been provided with a **makefile, LinkedList.cpp, LinkedList.h, main.cpp**.

Combine takes in a `LinkedList*` as a parameter. It combines that linkedlist with the current linked list and then sorts the list in ascending order and then returns a `LinkedList*` .

Ensure that if either of the lists is empty, the function still works.

If both lists are empty then return NULL.

The ordering should be case sensitive.

On completion, create a tarball containing all the source code. Upload it using the active fitchfork assignment called **Practical 8 - LinkedList**.

Task 2: Polymorphism [5]

Polymorphism is very powerful feature of the object orientated paradigm. The characteristic of polymorphism allows objects of a common type to be contained within a single container, allowing iterative calls to specialized methods through a common interface. In essence, the specialized type of an object need not be explicitly known to invoke an action.

This practical is a simple illustration of polymorphism's ability. You will create animal classes that can detect a human's presence and act accordingly. 5 small classes needs to be declared all inside the file "animals.h" and defined inside the file "animals.cpp". All member methods must be public. A simple main file is given to test your code. Make sure that you get the exact output, as fitch fork will enforce strict marking.

The five classes are Animal, Bird, TameAnimal, Dog, and Cat. All Constructors must have the signature (char const *) and should pass the parameter down to the highest abstracted class (i.e. Animal). A description of each class follows.

- `Class Animal:`
  An abstract class that will act as an interface, with a single protected member variable *'name'* that is of type `'char const *'`.

  Member methods follow:

  1. `Animal(char const * name_):`
     Constructor that initializes member variable *'name'*.
  2. `virtual char const * action()= 0;`
     Pure virtual method to be implemented in the concrete classes.
  3. `virtual void detectHuman():`
     Method that cout a line of the form "<name> detected a human and is <action()>".

- `Class Bird:`
  Inherits from Animal.

  Member methods follow:

  1. `virtual char const * action():`
     Method that returns "flying away".

- `Class TameAnimal:`
  Inherits from Animal.

  Member methods follow:

  1. `virtual void detectHuman():`
     Method that cout the line "<name> is looking for cuddles while <action()>".

- `Class Dog:`
  Inherits from TameAnimal.

  Member methods follow:

  1. `virtual char const * action():`
     Method that returns "wagging its tail".

- `Class Cat:`
  Inherits from TameAnimal.

  Member methods follow:

  1. `virtual char const * action():`
     Method that returns "purring loudly".

The following below are sample of the expected output:

```
***************** expected output *****************
Detections:
Dog is looking for cuddles while wagging its tail
Cat is looking for cuddles while purring loudly
Bird detected a human and is flying away

Actions:
wagging its tail
purring loudly
flying away
***************** expected output *****************
```

You are given the **main.cpp** and the **makefile**.

On completion, create a tarball containing all the source code. Upload it using the active fitchfork assignment called **Practical 8 - Polymorphism**.

Task 3: Least Common Multiple [5]

The least common multiple (LCM) of two numbers is defined as the smallest possible integer that is divisible by both numbers. The formula to do so is:

$$lcm(a,b) = \frac{|a*b|}{gcd(a,b)} \tag{1}$$

Where the greatest common divisor (GCD) is computed using the Euclidean algorithm:

$$gcd(a,0) = a \tag{2}$$

$$gcd(a,b) = gcd(b, a\%b) \tag{3}$$

Your task is to write a **recursive** function that can find the LCM between an arbitrary number of integers, with the prototype.

```
int lcm(int* numbers, int length);
```

Hint: `lcm(a, b, c)= lcm (a, lcm(b, c));`

It is advised that you make another recursive function to work out the gcd of two numbers.

Note that while it is possible to do these functions iteratively, this would require more work than programming them recursively. Additionally, this is a well-known problem in computer science and it would be beneficial to know how to go about programming it.

Students should implement the function in **lcm.cpp**.

On completion, create a tarball containing all the source code. Upload it using the active fitchfork assignment called **Practical 8 - Least Common Multiple**.