Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2018-2019

Programming Assignment 3

Due date: 28 April 2019, Monday, 23:59

1 Objectives

This homework aims to help you get familiar with the fundamental C++ programming concepts.

Keywords: *Constructor/Copy Constructor/Destructor, Assignment/Move, Operator Overloading, Memory Management*

2 Problem Definition

TL;DR: Implement the methods in the given 4 classes (Laptime, Car, Race, Championship). Cars have Laptimes, Races have Cars, Championships have Races. The details are on their way.

Another TA who watched Drive to Survive
...
Oh God show mercy to this boring soul.

F1 needs a new clear backbone for their visualisation system. As a Computer Engineering student from METU, they trusted you with the task. Your task is to keep track of the laptimes, cars, races and tracks. As situation requested shiny features of the latest C++ is not available to you. Therefore, you have to be careful with your programs memory management (You do not want to leave a program that is leaking memory as your legacy).

3 Class Definitions

3.1 Laptime

Laptime is the most basic class in this homework. Basically, it's the **node** in a **linked-list** which would keep the time information of the laps for each Car (not the best way to keep lap times of Cars, but let's assume F1 is not paying you anything and you let this go).

```
class Laptime {

private:
    int laptime;

    Laptime *next;

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PRIVATE METHODS/PROPERTIES BELOW

public:
    /**
     * Constructor.
     *
     * @param int value in laptime.
     */
    Laptime(int laptime);

    /**
     * Copy Constructor.
     *
     * @param rhs The laptime to be copied.
     */
    Laptime(const Laptime& rhs);

    ~Laptime();

    /**
     * Sets the next chain for this Laptime.
     *
     *
     * @param next The next Laptime.
     */
    void addLaptime(Laptime *next);

    /**
     * Less than overload.
     *
     * True if this Laptime less than the rhs Laptime.
     *
     * @param rhs The Laptime to compare.
     * @return True if this laptime is smaller, false otherwise.
     */
    bool operator<(const Laptime& rhs) const;

    /**
     * Greater than overload.
     *
```

```
     * True if this Laptime greater than the rhs Laptime.
     *
     * @param rhs The Laptime to compare.
     * @return True if this laptime is bigger, false otherwise.
     */
    bool operator>(const Laptime& rhs) const;

    /**
     * Indexing.
     *
     * Find the Laptime in desired position(start from zero).
     *
     * @return The Laptime with the given lap. Laptime with zero time if given ↩
        lap does not exists.
     */
    Laptime operator[](const int lap) const;

    /**
     *  Plus overload
     *
     *  Add two Laptime and return another Laptime
     *
     *  @param Laptime to add
     *  @returns Summation of the two laptime
     */
    Laptime& operator+(const Laptime& rhs);

    /**
     * Stream overload.
     *
     * What to stream:
     * minute:second.miliseconds
     *
     * Example:
     * 1:19.125
     *
     * @important Your laptime variable is representation in terms of miliseconds
     * and you have to turn it to desired outcome type
     *
     * @param os Stream to be used.
     * @param laptime Laptime to be streamed.
     * @return The current Stream.
     */
    friend std::ostream& operator<<(std::ostream& os, const Laptime& laptime);

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PUBLIC METHODS/PROPERTIES BELOW

};
```

3.2 Cars

Cars are similar to Laptimes, but they contain the name of the driver which is up to you (LeClerc can be a good start. Bahrain 2019 :/). It is again a **node** in a **linked-list**, but every car contains the **linked-list** of **Laptime** class (Memory Problems 2019).

```
#ifndef HW3_CAR_H
#define HW3_CAR_H

#include <ostream>
#include <vector>
#include <Laptime.h>

class Car {

private:
    std::string driver_name;
    double performance;
    Laptime *head;
    Car *next;

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PRIVATE METHODS/PROPERTIES BELOW

public:
    /**
     * Constructor.
     *
     * @Important: set the performance variable of the car by using Utilizer::↩
        generatePerformance()
     *
     * @param std::string The Car's driver name.
     */
    Car(std::string driver_name);

    /**
     * Copy Constructor.
     *
     * @param rhs The car to be copied.
     */
    Car(const Car& rhs);


    ~Car();
    /**
     * Gets the drivers name
     *
     *
     * @returns: drivers name
     */
    std::string getDriverName() const;

    /**
     *
     * Gets the performance
     *
     * @returns the performance
     *
     */
    double getPerformance() const;

    /**
     * Sets the next chain for this Car.
```

```
 * Adds a new car behind existing car
 *
 * Important: Car does NOT "own" next.
 *
 * @param next The next Car.
 */
void addCar(Car *next);


/**
 * Less than overload.
 *
 * True if total laptime of this Car is less than the rhs Car.
 *
 * Important:
 *
 * @param rhs The Car to compare.
 * @return True if this car's total laptime is smaller, false otherwise.
 */
bool operator<(const Car& rhs) const;

/**
 * Greater than overload.
 *
 * True if total laptime of this Car is greater than the rhs Car.
 *
 * Important:
 *
 * @param rhs The Car to compare.
 * @return True if this car's total laptime is greater, false otherwise.
 */
bool operator>(const Car& rhs) const;


/**
 * Indexing.
 *
 * Find the laptime of the given lap.
 *
 * @return The Laptime with the given lap. Laptime with zero time if given ↩
    lap does not exists.
 */
Laptime operator[](const int lap) const;

/**
 * Car completes one lap and records its laptime
 *
 * @Important: Based on your cars performance calculate some variance to add ↩
    average_laptime
 * Use  Utilizer::generateLaptimeVariance(performance) then add it to ↩
    average_laptime
 *
 * @param: Car takes average_laptime of the race
 *
 */
void Lap(const Laptime& average_laptime);
```

```
    /**
     * Stream overload.
     *
     * What to stream:
     * First Three letters of the drivers surname(Capitalized)--Latest Laptime--↩
       Fastest Laptime--Total Laptime
     * Example:
     * For Lewis Hamilton
     * HAM--1:19.235--1:18.832--90:03.312
     *
     * @Important: for lap numbers smaller in size you have to put zeros as much ↩
       as neccasary
     * @Important: you can use Laptime ostream when neccessary
     *
     * @param os Stream to be used.
     * @param car Car to be streamed.
     * @return The current Stream.
     */
    friend std::ostream& operator<<(std::ostream& os, const Car& car);

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PUBLIC METHODS/PROPERTIES BELOW

};
```

3.3 Race

Race keeps a **linked-list** of **Cars**. You have to keep your cars in the order from fastest to slowest (Nobody wants to see some randomly ordered car info). The details are in the code itself below:

```
class Race {

private:
    std::string race_name;
    Laptime average_laptime;
    Car *head;

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PRIVATE METHODS/PROPERTIES BELOW

public:
    /**
     * Constructor.
     *
     * @Important: Generate average_laptime by using Utilizer::↩
       generateAverageLaptime()
     *
     * @param int The year Race took place in.
     */
    Race(std::string race_name);
    /**
     * Copy Constructor.
     *
     * @Important just copy the names and performances of the car
     * without any laptime
     *
```

```
     * @param rhs The race to be copied.
     */
    Race(const Race& rhs);

    ~Race();

    int getRaceName() const;

    /**
     * Add a new car to race.
     *
     * @Important: At the start of the race their ordering is not important
     *
     * No parameter means that you will generate your own car in
     * this function(with a random name)and add it to your Cars
     */

    void addCartoRace();

    /**
     * Add a new car to race.
     *
     * @Important: At the start of the race their ordering is not important
     *
     * @param: car Add given Car to others
     */
    void addCartoRace(Car& car);

    /**
     * Information About how much car is in the race
     *
     * @returns number of cars
     *
     */
    int getNumberOfCarsinRace();

    /**
     * Return state of everything to desired lap's state
     *
     * @Important: this will also apply to cars and leaderboard too
     *
     * @param lap to return
     *
     */
    void goBacktoLap(int lap);


    /**
     * Prefix addition overload
     *
     * add one more lap to all cars
     *
     * @Important: Update the order of the cars so that the fastest one stays at ↩
        the front
     *
     */
    void operator++();
```

```
/**
 *  Prefix decrement overload
 *
 * remove one lap from all cars
 *
 * @Important: Update the order of the cars so that the fastest one stays at ↩
    the front
 *
 */
void operator--();

/**
 *
 * Indexing overload
 *
 *
 * @param: car_in_position Car in the given position
 * @returns the car in the desired position in the current lap
 */
Car operator[](const int car_in_position);

/**
 *
 * Indexing overload
 *
 * @param: driver_name driver's name of the desired car
 * @returns the car whose driver named as the given @param
 */
Car operator[](std::string driver_name);
/**
 * Assignment
 *
 * @param rhs The Race to assign into this race
 * @return The assigned Race
 */
Race& operator=(const Race& rhs);


/**
 * Stream overload.
 *
 * What to stream:
 * Position--*Driver Name(leader of the race)--Latest Laptime--Fastest ↩
    Laptime of the Driver--Sum of Laptimes(in display Laptime format)--Points↩
    --ExtraPoint(If applicable)
 * ...
 * Position--*Driver Name(last place of the race)--Latest Laptime--Fastest ↩
    Laptime of the Driver--Sum of Laptimes(in display Laptime format)--Points↩
    --ExtraPoint(If applicable)
 *
 * Example:
 * 001--TUF--1:19.461--1:18.935--60:35.193--25
 * 002--UTA--1:19.335--1:18.335--60:37.321--18--1
 * 003--GRT--1:20.223--1:19.932--60:45.184--15
 * ...
 * 099--CEI--1:21.005--1:19.867--63:47.293
```

```
     * 100--ECH--1:23.213--1:21.331--64:00.123
     *
     * @Important: for lap numbers smaller in size you have to put zeros as much as neccasary to their beginning
     * Example: if there is 11 racers first position should be 01
     * @Important. you can use Laptime ostream when neccessary
     * @Important: You should order the racers according to their total laptime
     * @Important: There are two different point types for F1
     * First one is the Fastest Lap point which is 1 point and it is given the fastest car if it is in top 10
     * Other one is normal racing points and they are 25-18-15-12-10-8-6-4-2-1 in this order
     *
     * @param os Stream to be used.
     * @param car Car to be streamed.
     * @return The current Stream.
     */
    friend std::ostream& operator<<(std::ostream& os, const Car& car);

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PUBLIC METHODS/PROPERTIES BELOW

};
```

3.4 Championship

Championship is the final part of this homework. They keep races with different names. Championship.

```
class Championship {

private:

    std::vector< Race > races;


    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PRIVATE METHODS/PROPERTIES BELOW

public:
    /**
     * Constructor.
     *
     * @param Championship name.
     */
    Championship();
    /**
     * Copy Constructor.
     *
     * @param rhs The Championship to be copied.
     */
    Championship(const Championship& rhs);

    ~Championship();


    /**
     * Add a new Race to Championship.
```

```
 *
 * @Important: You will use getAverageLaptime function for this
 * @Important: You can use copy constructor of the Race class in order
 * to just copy driver names for your new Race
 * @Important: Driver names should be same for each race
 *
 * @param race_name create a new race with given name
 */

void addNewRace(std::string race_name);

/**
 * Remove race from by using its name
 *
 * @param race_name remove race from championship
 *
 */
void removeRace(std::string race_name);




/**
 *
 * Indexing overload
 *
 *
 * @param: race name
 * @returns the desired Race
 */
Race operator[](std::string race_name);




/**
 * Stream overload.
 *
 * What to stream:
 * Championship Results
 * Driver Surname first three letters in capital(from winner)--Total Points
 * ...
 * Driver Surname first three letters in capital(to last place)--Total Points
 *
 * Example:
 * Championship Results
 * 01-RAI--194
 * 02-HAM--190
 * 03-LEC--100
 * 77-OCO--60
 * 78-RIC--1
 * 79-GRO--0
 *
 * @Important: for driver placements numbers smaller in size you have to put ↩
    zeros as much as neccasary to their start
 * Example: if there is 111 racers first position should be 001
 * @Important: You will order drivers according to their total points
```

```
     *
     * @param os Stream to be used.
     * @param car Car to be streamed.
     * @return The current Stream.
     */
    friend std::ostream& operator<<(std::ostream& os, const Championship& car);

    // DO NOT MODIFY THE UPPER PART
    // ADD OWN PUBLIC METHODS/PROPERTIES BELOW
};
```

4 Extras

While generating Cars you need to produce random double value for its performance. To do this, you **MUST** use Utilizer class. It'll return random small double numbers.

The implementation of the the Utilizer is already provided to you. Hence, you just need to do this:

```
double performance = Utilizer::generatePerformance();
```

You also **MUST** use Utilizer::generateLaptimeVariance() for generating a difference to average laptime. After that you will add this to calculate laptime of the car (You will do this for every lap). For generating average_laptimes of the Races you again **MUST** use Utilizer::generateAverageLaptime() function which will return an integer value (You will turn it to Laptime class).

The summary of the memory ownership:

- A Laptime **WILL NOT** own its `nextLaptime`.
- A Car **WILL NOT** own its `nextCar`.
- A Car **WILL** own the `headLaptime` when constructed with.
- A Race **WILL** own the `headCar`.
- CopyConstructers **WILL** not yield ownership of the old variables.
- assignment operator "=" **WILL** yield ownership of the old variables.
- Owning a `Car/Laptime` also means owning the `nextCar/nextLaptime`, and the `nextCar/nextLaptim` of the `nextCar/nextLaptime`, ...

5 Grading

- Full grade for Laptime class implementation **15** points.
- Full grade for Car class implementation **30** points.
- Full grade for Race class implementation **30** points.
- Full grade for Championship class implementation **25** points.

In order to get full grade from each part your code should not have any memory leak. This will be checked with valgrind. While grading your classes will be used with the correct implementations, therefore they are expected to work as commented in the code.

6 Regulations

- **Programming Language:** You must code your program in C++ (11). Your submission will be compiled with `g++` with `-std=c++11` flag on department lab machines.
- **Allowed Libraries:** You may include and use C++ Standard Library. Use of any other library (especially the external libraries found on the internet) is forbidden.
- **Memory Management:** When an instance of a class is destructed, the instance must free all of its owned/used heap memory. Any heap block, which is not freed at the end of the program will result in grade deduction. Please check your codes using valgrind –leak-check=full for memory-leaks.
- **Late Submission:** You have a total of 10 days for late submission. You can spend this credit for any of the assignments or distribute it for all. For each assignment, you can use at most 3 days-late.
- **Cheating:** In case of cheating, the university regulations will be applied.
- **Newsgroup:** It's your responsibility to follow the cengclass forums for discussions and possible updates on a daily basis.

7 Submission

Submission will be done via CengClass. Create a zip file named `hw3.zip` that contains:

- Laptime.h
- Laptime.cpp
- Car.h
- Car.cpp
- Race.h
- Race.cpp
- Championship.h
- Championship.cpp

Do not submit a file that contains a `main` function. Such a file will be provided and your code will be compiled with it. Also, do not submit a Makefile.

Note: The submitted zip file should not contain any directories! The following command sequence is expected to run your program on a Linux system:

```
$ unzip hw3.zip
$ make clean
$ make all
$ make run
$ -optional- make valgrind
```