UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 2 Specifications: Classes with Dynamic Memory, Copy, Assignment, Destructor

Release date: 14-08-2023 at 06:00

Due date: 18-08-2023 at 23:59

Total Marks: 282

Contents

1 General Instructions: **2**

2 Plagiarism **3**

3 Outcomes **3**

4 Introduction **3**

5 Class Diagram **3**

6 Classess **4**

6.1 Accessory 4

6.2 Engine 5

6.3 Car 6

7 Memory Management **9**

8 Testing **9**

9 Upload checklist **9**

10 Submission **10**

1 General Instructions:

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- The following imports are allowed ("<iostream>", "<sstream>", "<string>" ,"<stdlib.h>", "engine.h", "car.h", "accessory.h"). You do not need all of these, however any imports part from these will result in a mark of 0.
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**

- We will be overriding your .h files, therefore do not add anything extra to them that is not in the spec
- In your student files you have been provided all the files you will need, however they are mostly empty and you will need to implement them.

2 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 material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The goal of this practical is to gain experience with working with dynamic classes, using the copy constructor and overloading the assignment operator.

4 Introduction

Implement the UML diagram and functions as described on the following pages.

5 Class Diagram

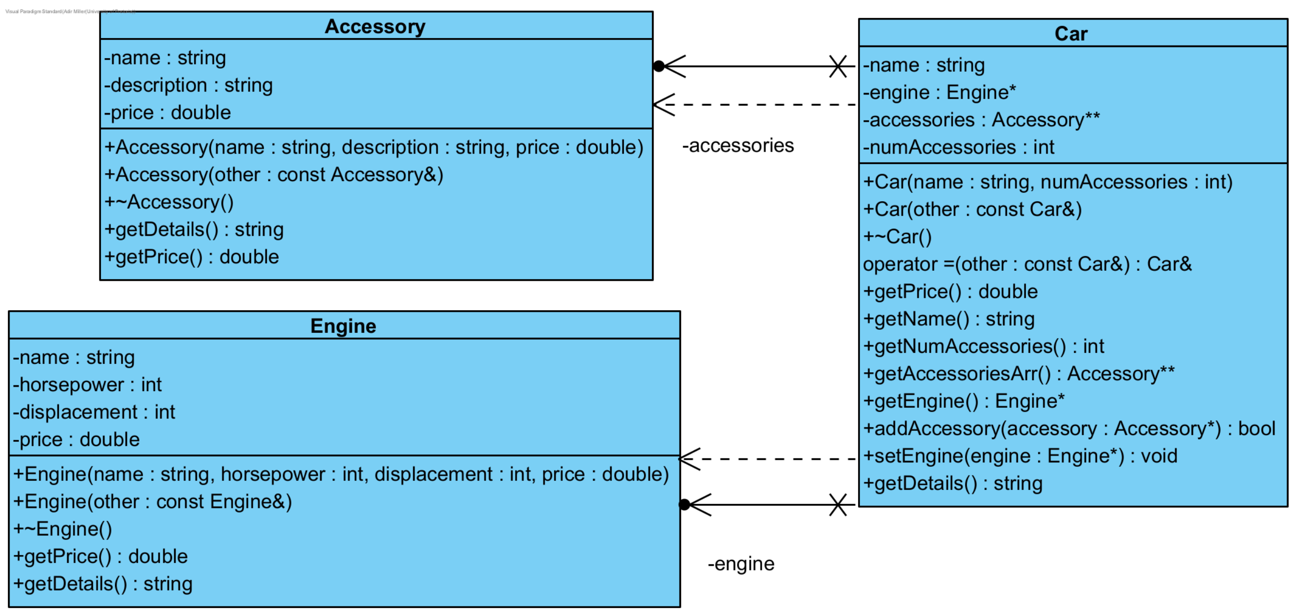
Accessory
-name : string
-description : string
-price : double
+Accessory(name : string, description : string, price : double)
+Accessory(other : const Accessory&)
+~Accessory()
+getDetails() : string
+getPrice() : double
-accessories
Car
-name : string
-engine : Engine*
-accessories : Accessory**
-numAccessories : int
+Car(name : string, numAccessories : int)
+Car(other : const Car&)
+~Car()
operator =(other : const Car&) : Car&
+getPrice() : double
+getName() : string
+getNumAccessories() : int
+getAccessoriesArr() : Accessory**
+getEngine() : Engine*
+addAccessory(accessory : Accessory*) : bool
+setEngine(engine : Engine*) : void
+getDetails() : string
Engine
-name : string
-horsepower : int
-displacement : int
-price : double
+Engine(name : string, horsepower : int, displacement : int, price : double)
+Engine(other : const Engine&)
+~Engine()
+getPrice() : double
+getDetails() : string
-engine

Figure 1: Class diagrams

6 Classess

6.1 Accessory

- Members:
  - name: string
    - Simple member variable which stores the name of the accessory.
    - The variable is initialized in the Accessory constructor.
  - description: string
    - Simple member variable which stores the description of the accessory.
    - The variable is initialized in the Accessory constructor.
  - price: double
    - Simple member variable which stores the description of the accessory.
    - The variable is initialized in the Accessory constructor.
- Functions:
  - Accessory(name: string, description: string, price: double)
    - The is the constructor for the Accessory class.
    - Initializes the *name* member variable with the name parameter.
    - Initializes the *description* member variable with the description parameter.
    - Initializes the *price* member variable with the price parameter.
  - ∼Accessory()
    - This is the destructor for the Accessory class.
    - It does not deallocate anything, but is required for FitchFork.
  - Accessory(other: const Accessory&)
    - Copy constructor for Accessory class.
    - Initializes all member variables with values from the Accessory object passed in.
  - getDetails(): string
    - This is a getter function.
    - Note that "{{foo}}" is not C++ syntax, it is simply an indication you should **replace** it with the corresponding member variable.
    - This function should return a string in the following format:

```
Accessory Name: {{name}}\n
Description: {{description}}\n
Price: {{price}}
```

    - **NB**: only the first **2 lines** should have a linebreak (\n) at the end.
    - The price variable will need to be parsed to a string before it can be added to the string to be returned. Use the stringstream library for this.
    - This function only returns a string, it does not print anything.
  - getPrice(): double
    - A simple getter that returns the value of the member variable *price*.

6.2 Engine

- Members:
  - name: string
    - Simple member variable which stores the name of the engine.
    - The variable is initialized in the Engine constructor.
  - horsepower: int
    - Simple member variable which stores the horsepower of the engine.
    - The variable is initialized in the Engine constructor.
  - displacement: int
    - Simple member variable which stores the displacement of the engine.
    - The variable is initialized in the Engine constructor.
  - price: double
    - Simple member variable which stores the price of the engine.
    - The variable is initialized in the Engine constructor.
- Functions:
  - Engine(name: string, horsepower: int, displacement: int, price: double)
    - This is the constructor of the Engine class.
    - Initializes the *name* member variable with the name parameter.
    - Initializes the *horsepower* member variable with the horsepower parameter.
    - Initialize the *displacement* member variable with the displacement parameter.
    - Initialize the *price* member variable with the price parameter.
  - Engine(other: const Engine&)
    - Copy constructor for Engine class.
    - Should initialize all member variables with values from the Engine object passed in.
  - ~Engine()
    - The destructor for the engine class.
    - It does not deallocate anything, but is required for FitchFork.
  - getDetails(): string
    - This is a getter function.
    - Note that "{{foo}}" is not C++ syntax, its simply an indication you should replace it with the corresponding member variable.
    - This function should return a string in the following format:

```
Engine name: {{name}}\n
Horsepower: {{horsepower}}\n
Displacement: {{displacement}}\n
Price: {{price}}
```

    - **NB**: only the first **3 lines** should have a linebreak (\n) at the end.
    - The price, displacement, and horsepower variables will need to be parsed to a strings before it can be added to the string that needs to be returned. Use the stringstream library for this.
    - This function only returns a string, it does not print anything.
  - getPrice(): double
    - A simple getter that returns the value of the member variable *price*.

6.3 Car

- Members:
  - name: string
    - Simple member variable which stores the name of the car.
    - The variable is initialized in the Car constructor.
  - numAccessories: int
    - Simple member variable which stores the number of accessories a car can have.
    - This is used as the max size of the *accessories* array.
    - The variable is initialized in the Car constructor.
  - engine: Engine*
    - A pointer to an Engine object.
    - This variable is initialized to **NULL**[1] in the Car constructor.
    - Once this pointer is pointing to an object set by *setEngine()*, then the Car object is responsible for the deletion (deallocation) of that object.
  - accessories: Accessory**
    - This is a **1D array of dynamic objects**.
    - The array should be made during construction and properly initialized.
    - The size of the array is determined by the *numAccessories* variable.
- Functions:
  - Car(name: string, numAccessories: int)
    - The constructor for the Car class.
    - Initialize the *name* member variable with the name parameter.
    - Initializes the *numAccessories* member variable with the numAccessories parameter.
    - The *accessories* array should be initialized with a size of *numAccessories*.
    - If numAccesssories is 0, then the *accessories* array should be set to *NULL*.
    - If the *numAccesssories* is a negative number, it should be set to 0 and the *accessories* array should be set to *NULL*.
    - The *engine* member variable should be set to *NULL*.
  - Car(const Car& other)
    - This is the copy constructor for the Car class.
    - When copying over from the other Car object, ensure that you use a **deep copy**.
  - ~Car()
    - This is the destructor for the Car class.
    - All elements of the *accessories* array that are **not** *NULL*, must be deleted.
    - If the *accessories* array is **not** *NULL*, it must be deleted.
    - If an engine is set, it must be deleted.
  - operator=(other: const Car&): Car&
    - The assignment operator needs to be overloaded.
    - The object should be restructured in such a way it matches the object that was passed in as a parameter.

[1]Remember we use *NULL*, not *nullptr* since *nullptr* is a C++11 feature

    - Remember to only use **deep copies**.
    - If the Car object is storing any pointers to the *accessories* array or Engine object, the accessories array (as well as any accessories it may be storing) and the Engine object needs to be deallocated **before** the Car object makes deep copies of the accessories array and the Engine object passed in.
    - Ensure that an object cannot assign itself, to itself.
    - Ensure that your code is capable of chaining.
- getPrice(): double
    - This is a getter function.
    - This should return the sum of the price of the engine and all accessories combined.
    - Since it is not guaranteed that the engine is set or accessories have been added, ensure that this function takes this condition into account.
    - If there is no engine and no accessories, then simply return 0.
- getName(): string
    - This is a getter function.
    - Returns the value of the *name* member variable.
- getNumAccessories(): int
    - This is a getter function.
    - Returns the value of the *numAccessories* member variable.
- getAccessoriesArr(): Accessory**
    - This is a getter function.
    - Returns the pointer to the accessories array stored in the *accessories* member variable.
    - If *accessories* is *NULL*, this method should return *NULL*.
- getEngine(): Engine*
    - This is a getter function.
    - Returns the pointer stored in the *engine* member variable.
    - If *engine* is *NULL*, this method should return *NULL*.
- addAccessory(accessory: Accessory*): bool
    - This method is used to add an accessory to the *accessories* array.
    - If an object can be added then *true* should be returned, otherwise return *false*.
    - To determine if an object can be added, is dependent on the size of the *accessories* array, *numAccessories*, and how many spaces are left open. To determine how many spots are left open in the array, you must loop through the array and determine how many elements are still *NULL*, this is also how you will find out where to place the new object in the array.
    - You must add elements to the array in a natural order, i.e. you cannot first add to the end, then to the beginning, then to the middle of the array.
    - You may assume that each accessory that is added is unique.
    - If the array is *NULL* (*numAccessories*=0) then just return false.
    - During destruction, you are only responsible for deleting elements that have been added to the array, if this method fails to add to the array and returns *false*, then it is the method caller's responsibility to delete that object.
    - Assume if an accessory is added to one car, it won't be added to another car.
- setEngine(engine: Engine*): void
    - This method is used to set the *engine* member variable.

    * Once an engine is set, you cannot set it back to *NULL*, meaning that if an engine is set and *NULL* is passed in then nothing should happen.
    * An engine can be changed if a new engine is passed in, then the current must be deleted and the new one set to *engine*.
    * If the engine that is currently set is passed in again, nothing should happen. This means:

    ```
    Engine* testEngine=...;// assume a valid engine is created
    Car* testCar=...;// assume a valid car is created
    testCar->setEngine(testEngine);// set the engine
    testCar->setEngine(NULL);// has no effect
    testCar->setEngine(testEngine);// has no effect
    ```

    * Assume if an engine is set for one car, it won't be set to another.
    * Remember once an Engine is set, it is the cars object's responsibility to delete it.
- getDetails(): string
    * This is a getter function.
    * Note that "{{foo}}" is not C++ syntax, its simply an indication you should replace it with the corresponding member variable.
    * This method will return a string, it will always begin with the following:

    ```
    Details:\n
    Car: {{name}}\n
    Price: {{getPrice()}}
    ```

    * If an engine is set, the result of *engine->getDetails();* should be appended.
    * If there is an accessories array then you should loop through and call *getDetails()* on the objects in the array, note **only** call *getDetails()* on objects that exist in the array. This means that even if *numAccessories* is 4 but you only have added 2 Accessories to the array then only 2 accessories should be appended.
    * **NB** Do **not** end the string with a newline.
    * This function only returns a string, it does not print anything.
    * To make this a bit clearer, here are a few examples of the final string. Pay close attention to **newlines**:

    ```
    //Car with 2 accessories and an engine
    Details:\n
    Car: {{Foo}}\n
    Price: {{Foo}}\n
    Engine name: {{Foo}}\n
    Horsepower: {Foo}\n
    Displacement: {{Foo}}\n
    Price: {{Foo}}\n
    Accessory Name: {{Foo1}}\n
    Description: {{Foo1}}\n
    Price: {{Foo1}}\n
    Accessory Name: {{Foo2}}\n
    Description: {{Foo2}}\n
    Price: {{Foo2}}
    ```

    ```
    //Car with no engine and 1 accessory
    Details:\n
    Car: {{Foo}}\n
    Price: {{Foo}}\n
    Accessory Name: {{Foo1}}\n
    Description: {{Foo1}}\n
    Price: {{Foo1}}
    ```

7 Memory Management

Memory management is a core part of COS110 and C++, so this practical memory management is extremely important due to the scope. Therefore each task on FitchFork will allocate approximately 20% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

8 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the practical marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the main.cpp file) that will be used to test the Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing, the gcov [2] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j accessory engine car
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark	Final mark
0%-5%	0%	0
5%-20%	20%	6
20%-40%	40%	12
40%-60%	60%	18
60%-80%	80%	24
80%-100%	100%	30

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the Instructor Provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

9 Upload checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

[2]For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

- main.cpp
- engine.cpp
- accessory.cpp
- engine.cpp

Ensure that all files are in lowercase. You will notice you do not need to upload a makefile or any h files, you may if you wish, however, FF will overwrite them before running your submission.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

10 Submission

You need to submit your source files, only the cpp files (**including the main.cpp**), on the FitchFork website (https://ff.cs.up.ac.za/). All methods must be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or .h files in your submission. **Ensure your files are in the root directory of the zip file**. Your code should be able to be compiled with the C++98 standard
For this practical you will have 10 upload opportunities. Upload your archive to the Practical 2 slot on the Fitch Fork website.