CMP2801M Workshop Week 7

FUNCTION OVERLOADING AND COPY CONSTRUCTORS

Write and test your code, and add comments so that you can refer to it later (focus on *why* you made a particular decision to do something a certain way). This will help you to understand your work when you refer to it in future. Please refer to the lecture notes or ask for help if needed! There are two parts to this workshop: **I)** looking back to the last topic, for review and practice of Inheritance, and **II)** exploring the fundamentals of this week's topic, Function Overloading and Copy Constructors.

PART I – Inheritance

Task 1:

Given the following console output, implement an appropriate class hierarchy to produce the following console output, using the provided `main()`, without changing it. You may only add a single `cout` statement (one line each) to each of your constructors and destructors.

```
int main (void)
{
      ClassChild c;
      cout << c.hello() << endl;
      return 0;
}
```

Desired console output:

```
      > class instance created
      > class instance created
      > ChildClass instance created
      > Hello!
      > ChildClass instance deleted
      > class instance destroyed
      > class instance destroyed
```

Task 2:

Based on your implementation of the Robot class, with the extensions added, from week 6, create a new class of robot named `AutonomousCar`, using inheritance. Add the following properties to this new class (with further additions/modifications as you see fit):

1. A constructor that allows the name of the car to be set (consider what property from which class may be the most appropriate for this),
2. Appropriate private properties that can store the current location of the vehicle in Cartesian coordinates, and update this when required,

3. A new method (choose an appropriate name) that does the following: (a) takes two arguments, distance and orientation, (b) calculates a new location based on the inputs, and the current location, (c) updates the current position of the vehicle, (d) prints to console what has happened.
4. An override of the `Robot` class `work()` method, that informs the user that the `AutonomousCar` does not work, but drives!

Task 3:

Refer to the source file `workshop7-task3.cpp`. Implement the `Tool` class. It should have an `int` property called `strength` and a `char` property called `type`. Choose whether private or protected is more appropriate. This `Tool` class should also contain the method `setStrength(int)`, which sets the `strength` for the `Tool.`

Create three more classes as labelled in the provided source file: `Rock`, `Paper`, and `Scissors`, each of which inherits from `Tool`. Each of these classes requires a constructor that takes an `int` argument used to initialise the `strength` property. The constructor should also initialise the `type` property: 'r' for `Rock`, 'p' for `Paper,` and 's' for `Scissors`. Ensure each of the classes is appropriately split into header and source files.

These classes will also need a public method `fight(Tool)` that compares their `strengths` in the following way:

- `Rock`'s `strength` is doubled (temporarily) when fighting `Scissors`, but halved (temporarily when fighting `Paper`.
- In the same way, `Paper` has the advantage over `Rock`, and `Scissors` against `Paper`.
- The `strength` property shouldn't change in this method, which returns `true` if the original class wins in `strength`, and `false` otherwise.

Run the program without changing the main function, and verify that the results are as expected.

PART II – Function Overloading and Copy Constructors

For this part of the workshop, refer to the reading material on Blackboard "reading-week7.pdf". This will provide an overview/preview of the topic for the week 7 lecture: Function Overloading and Copy Constructors. It would be worth also recapping the content on Memory/Memory Management (week 3) as part of your preparation for the topic this week.

Task 4:

Create a program that implements the following functions, along with a `main()` function that can test these in a suitable manner. Ensure that all functions are explicitly tested, verify whether your program operates as expected, and read through all compiler warnings and errors. Each of the following functions should take two arguments and return their sum:

1. `int add (int, int)`
2. `float add (int, int)`
3. `float add (float, int)`
4. `float add (int, float)`
5. `float add (float, float)`

Note down (in the comments of the code for example) what the warnings/errors are and what they refer to. Is this expected behaviour? Comment out any lines/functions that cause errors once you have noted what these errors are.

Task 5:

(This task is primarily for exploration – please use the following points as a starting point, but explore further changes as you see fit. After any changes, re-run your program and assess the following: is this expected behaviour? And, what do the compiler warnings and errors tell you?)

Take the Robot class (from Task 2, above, for example) and extend it in the following ways:

- Add private member property (pointer to vector): `vector<int>* _history`
- In the class constructor, allocate memory this pointer (using `new`). Ensure the destructor is appropriately updated too.
- Add a "setter" method for the `name` property of the class: `void setName(string)`.
- Whenever the `work()` method is called, then add the new value of `workUnit` to the end of `_history`.
- Add a method to the `Robot` class that prints out the contents of the `_history` vector to console: `void printWork()`

Use the following `main()` function as a starting point, but extend as desired to explore whether the expected behaviours occur. Add comments to this code as you explore it.

```
int main (void)
{
	Robot r2;
	{
		Robot r1("r1");
		r1.whoAmI();
		r1.work();
		r1.work();
		r1.printWork();
		cout << "assign r1 to r2..." << endl;
		r2 = r1;
		r2.setName("r2");
		r2.whoAmI();
		r2.printWork();
	}
	r2.whoAmI();
	r2.printWork();
	cout << "end of example code..." << endl;
	return 0;
}
```

Note down what is going on here. Is this what you expect to happen? What actually happens?