CMP2801M Workshop Week 11

POLYMORPHISM

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 Virtual Functions, and **II)** exploring the fundamentals of this week's topic, Polymorphism.

PART I – Virtual Functions

Task 1:

Fix the following code to produce the desired output in main (*without* changing the body of main!). It would help to implement this – use the error messages when you try to compile to help find the problems.

```
    class A                                               class B : public A
    {                                                     {
        void report (void) {                                  void report (void) {
            cout << “I am parent class” << endl;                  cout << “I am child class” << endl;
        }                                                     }
    }                                                     }

    void main ()
    {
        //don’t change any of the following
        B b;
        A *aP = &b;
        cout << aP->report() << endl;    // want this to produce “I am child class”
        return 0;
    }
```

Task 2:

Implement (or update your implementation from the lecture in week 6, and/or the workshop in week 7) the `Robot` class covered in the lecture this week, and also implement the two derived classes `SocialRobot` and `IndustrialRobot` (properly separated into separate declaration/definition files).

A. Implement a pure virtual `sayHello` method in the Robot class, and appropriately update the two derived classes.
B. Update `Robot` as an *Interface class*. Do you need to change anything in your two derived classes?

Task 3:

In your `main` function, create a container for pointers to objects of type `Robot` (consider a `vector` for example), named `robotContainer`. Populate this vector with pointers to instances of both `SocialRobot` and `IndustrialRobot` (at least one of each). Write a function (separate from any of the class definitions) that calls the `sayHello` method for all elements in `robotContainer`, ensuring that the appropriate behaviour is executed for each object type, and that the function is efficient for memory use.

A. If this vector stored objects of type `Robot` (not pointers), what would happen?

Task 4:

Implement a virtual destructor for the `Robot` inheritance hierarchy, and verify that it is working as expected for objects of both derived classes (i.e. destructors for both the derived class and the base class are called appropriately) in the `robotContainer`. Include print outs in the destructors to demonstrate what is happening.

PART II – Polymorphism

In this part of the workshop, we preview/review (depending on whether you have seen the lecture yet this week or not) the topic for week 11 (Polymorphism).

Task 5:

Given the following code, update such that the console output shows the contents of the C class destructor.

```
class A
{

};
```

```
class B : public A
{

};
```

```
class C : public B
{
  ~C() {
    cout << "C destructor" << endl;
  }
};
```

```
int main()
{
        A *a = new C;
        delete a;
        return;
}
```

Task 6:

Given the following function, implement the inheritance hierarchy and contents of main to be able to use it: verify that it works on objects from three different levels of the inheritance hierarchy. Note that the vector <u>out</u> should be empty when passed to this function, but the vector <u>in</u> should be populated.

```
bool Function(vector<A*> in, vector<A*> &out)
{
        for (int i = 0; i < (int)in.size(); i++)
        {
                A *ptr;
                if (in[i]->name() == "A") {
                        ptr = new A;
                }
                else if (in[i]->name() == "B") {
                        ptr = new B;
                }
                else if (in[i]->name() == "C") {
                        ptr = new C;
                }
                else {
                        cout << "What are you trying to do?!" << endl;
                        return false;
                }
                out.push_back(ptr);
                out[i]->sayHello();
        }
        return true;
}
```

Consider the following given the code above, and the code that you have written:

A. Determine **three ways** in which the provided function could be made **more efficient** (there may be further ways to improve efficiency…)

B. An alternative to implementing a `name()` member function would be to use `typeid()`, for which the header `<typeinfo>` should be included – try to use this instead.

Additional:

If you have gotten this far, then take the opportunity to complete the week 9 workshop if not already done. For further challenges, consider the following:

A. https://www.hackerrank.com/challenges/virtual-functions/problem

B. https://www.hackerrank.com/challenges/abstract-classes-polymorphism