CptS 122 – Data Structures

Lab 13: Polymorphism in C++

Assigned: Monday, July 24, 2017
Due: At the end of the lab session

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement and test classes in C++ which apply polymorphism
- Apply inheritance and polymorphism to model and simulate Animals and Pets

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Create test cases for a program
- Design, implement and test classes in C++ which apply inheritance
- Compare and contrast inheritance ("is-a") relationships versus composition ("has-a") relationships
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*
- Describe and define *inheritance*

III. Overview & Requirements:

This lab, along with your TA, will help you navigate through designing, implementing, and testing polymorphism with classes in C++. It will also, once again, help you with understanding how to apply inheritance to an application.

Labs are held in a "closed" environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Please help other students in need when you are finished with a task. You may work in pairs if you wish. However, I encourage you to compose your own solution to each problem. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

Tasks:

NOTE: Parts of this lab are courtesy of Jack Hagemeister.

Please work in your teams for this lab!!!!

Task 1.

To gain a better understanding of polymorphic and virtual functions start with the following simple example. Notice we have not defined a virtual function yet.

```
// Task1.h

#include <iostream>

using std::cout;
using std::endl;

class Base
{
      public:
            void testFunction ();
};

class Derived : public Base
{
      public:
            void testFunction ();
};

// Task1.cpp

#include "Task1.h"

void Base::testFunction ()
{
    cout << "Base class" << endl;
}

void Derived::testFunction ()
{
    cout << "Derived class" << endl;
}

// main.cpp

#include "Task1.h"

int main(void)
{

    Base* ptr = new Base;

    ptr -> testFunction ();         // prints "Base class"

    delete ptr;

    ptr = new Derived;

    ptr -> testFunction ();         // prints "Base class" because the base class function
is not virtual

    delete ptr;
```

```
    return 0;
}
```

Now modify the code with the following (all other code should remain the same).

```
class Base
{
      public:
            virtual void testFunction ();
};
```

Compile and run your program with this modification. You'll notice the second `testFunction()` call generates the message "Derived class". Welcome to polymorphism!

Task 2.

You will first build two classes, Mammal and Dog. Dog will inherit from Mammal. Below is the Mammal class code. Once you have the Mammal class built, build a second class Dog that will inherit publicly from Mammal.

```
// Mammal.h

#pragma once

#include <iostream>

using std::cout;
using std::endl;

class Mammal
{
      public:
            Mammal(void);
            ~Mammal(void);

            virtual void Move() const;
            virtual void Speak() const;

      protected:
            int itsAge;
};

// Mammal.cpp

#include "Mammal.h"

Mammal::Mammal(void):itsAge(1)
{
      cout << "Mammal constructor..." << endl;
}

Mammal::~Mammal(void)
{
      cout << "Mammal destructor..." << endl;
}

void Mammal::Move() const
{
      cout << "Mammal moves a step!" << endl;
}
```

```
void Mammal::Speak() const
{
      cout << "What does a mammal speak? Mammilian!" << endl;
}
```

Once you have completed class Mammal and Dog, build the following main program.

```
#include "Mammal.h"
#include "Dog.h"

int main ()
{
   Mammal *pDog = new Dog;

   pDog->Move();
   pDog->Speak();

   //Dog *pDog2 = new Dog;

   //pDog2->Move();
   //pDog2->Speak();

   return 0;
}
```

What does it output, is that what you expected? Remove the keyword virtual from the class mammal and try it again. Now what happens? Next, put in another pointer to pDog2 in the main program, but this time make it a pointer to a Dog, not a mammal and create a new dog. Now what happens? What you should realize is that by making the method Speak virtual, we can have a little different behavior through dynamic (runtime) binding.

Task 3.

Develop additional classes for Cat, Horse, and GuineaPig overriding the move and speak methods. (If you do not know guinea pigs go “weep weep”)

Next, test with the modified main:

```
int main ()
{
   Mammal* theArray[5];
   Mammal* ptr;
   int choice, i;
   for (i = 0; i<5; i++)
   {
      cout << "(1)dog (2)cat (3)horse (4)guinea pig: ";
      cin >> choice;
      switch (choice)
      {
         case 1: ptr = new Dog;
         break;
         case 2: ptr = new Cat;
         break;
         case 3: ptr = new Horse;
         break;
         case 4: ptr = new GuineaPig;
         break;
         default: ptr = new Mammal;
         break;
      }
```

```
        theArray[i] = ptr;
    }
    for (i=0;i<5;i++)
        theArray[i]->Speak();
    return 0;
}
```

Some things to note:

If the Dog object had a method, WagTail(), which is not in the Mammal, you could not use the pointer to Mammal to access that method (unless you cast it to be a pointer to Dog). Because WagTail() is not a virtual function, and because it is not in a Mammal object, you can't get there without either a Dog object or a Dog pointer to the Dog object!!!

The virtual function magic (polymorphic behavior) operates only on pointers and references. Passing an object by value will not enable the virtual functions to be invoked.

Some questions that you should understand:

Are inherited members and functions passed along to subsequent generations? If Dog derives from Mammal, and Mammal derives from Animal, does Dog inherit Animal's functions and data?

A. Yes. As derivation continues, derived classes inherit the sum of all the functions and data in all their base classes.

Q. If, in the example above, Mammal overrides a function in Animal, which does Dog get, the original or the overridden function?

A. If Dog inherits from Mammal, it gets the function in the state Mammal has it: the overridden function.

Q. Can a derived class make a public base function private?

A. Yes, and it remains private for all subsequent derivations.

Q. Why not make all class functions virtual?

A. There is overhead with the first virtual function in the creation of a v-table. After that, the overhead is trivial. Many C++ programmers feel that if one function is virtual, all others should be. Other programmers disagree, feeling that there should always be a reason for what you do.

Q. If a function (SomeFunc()) is virtual in a base class and is also overloaded, so as to take either an integer or two integers, and the derived class overrides the form taking one integer, what is called when a pointer to a derived object calls the two-integer form?

A. The overriding of the one-int form hides the entire base class function, and thus you will get a compile error complaining that that function requires only one int.

Here are some more questions:

1. What is a v-table?

2. What is a virtual destructor?

3. How do you show the declaration of a virtual constructor?

4. How can you create a virtual copy constructor?

5. How do you invoke a base member function from a derived class in which you've overridden that function?

6. How do you invoke a base member function from a derived class in which you have not overridden that function?

7. If a base class declares a function to be virtual, and a derived class does not use the term virtual when overriding that class, is it still virtual when inherited by a third-generation class?

8. What is the protected keyword used for?

Some more exercises:

1. Show the declaration of a virtual function that takes an integer parameter and returns void.

2. Show the declaration of a class Square, which derives from Rectangle, which in turn derives from Shape.

3. If, in Exercise 2, Shape takes no parameters, Rectangle takes two (length and width), but Square takes only one (length), show the constructor initialization for Square.

4. Write a virtual copy constructor for the class Square (in Exercise 3).

5. BUG BUSTERS: What is wrong with this code snippet?

```
void SomeFunction (Shape);
Shape * pRect = new Rectangle;
SomeFunction(*pRect);
```

6. BUG BUSTERS: What is wrong with this code snippet?

```
class Shape() { public: Shape(); virtual ~Shape(); virtual Shape(const Shape&); };
```

IV. Submitting Labs:

- You are not required to submit your lab solutions. However, you should keep them in a folder that you may continue to access throughout the semester. You should not store your solutions to the local C: drive on the EME 120/128 machines. These files are erased on a daily basis.

V. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time and continue to work on the problems until the TA has dismissed you.