Lab 12

Polymorphism

Objective: After completion of this lab, you will be able to

- create classes in C++ which apply polymorphism.

Lab Exercise

Part 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.

```
// Part1.h. Note, this header file will have TWO classes declared.

#include <iostream>

using namespace std;

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

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


// Part1.cpp. Note this implementation file will have TWO class defined.

#include "Part1.h"

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

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

```
// main.cpp
#include "Part1.h"

int main()
{

    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! No need to submit anything for this part of the lab, just make sure that **`virtual`** is clear to you.

Part 2.

You will 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`**. The **`Dog`** class should also override the **`move()`** and **`speak()`** methods from **`Mammal`**.

```
// Mammal.h

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

            virtual void move() const;
```

```
            virtual void speak() const;

      protected:
            int itsAge;
};

// Mammal.cpp
#include "Mammal.h"

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



Mammal::~Mammal()
{
      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();

   delete pDog;
   //delete pDog2;

   return 0;
}
```

Part 3.

Develop additional classes for **`Cat`**, **`Horse`**, and **`GuineaPig`** overriding the **`move()`** and **`speak()`** methods. (If you didn't know, guinea pigs go "wheep wheep"). Make sure to also define a constructor and destructor for each of these classes, and that when you run your program, you can also see that the correct constructors and destructors are being called. Recall that a derived class will always call the parents constructors and destructors, so if you see this behavior, that is okay. *Just make sure you are also calling the derived class' constructors and destructors.*

Test with the modified main:

```
int main ()
{
   Mammal* theArray[5];
   Mammal* ptr;
   int choice;
   for (int 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;
   }

   // Iterate through array, and have each animal speak
   for (int i = 0; i < 5; i++)
   {
      theArray[i]->speak();
   }

   // Always free dynamically allocated objects
   for (int i = 0; i < 5; i++)
   {
```

```
        delete theArray[i];
    }

    return 0;
}
```

Sample Run:

```
(1)dog (2)cat (3)horse (4)guinea pig: 1
Mammal constructor...
Dog constructor...
(1)dog (2)cat (3)horse (4)guinea pig: 2
Mammal constructor...
Cat constructor...
(1)dog (2)cat (3)horse (4)guinea pig: 3
Mammal constructor...
Horse constructor...
(1)dog (2)cat (3)horse (4)guinea pig: 4
Mammal constructor...
Guinea Pig constructor...
(1)dog (2)cat (3)horse (4)guinea pig: 5
Mammal constructor...
What does a dog say? Woof!
What does a cat say? Meow!
What does a horse say? Ney!
What does a Guinea Pig say? Wheep WHEEP!
What does a mammal speak? Mammilian!
Dog destructor...
Mammal destructor...
Cat destructor...
Mammal destructor...
Horse destructor...
Mammal destructor...
Guinea Pig destructor...
Mammal destructor...
Mammal destructor...
```