Fpt Software	**CODE:** CPP.Assignment08.Opt2 **Assignment topic:** Advance OOP, Inheritance and Polymorphism **Assignment duration:** 180 minutes	**FRESHER ACADEMY**

Bài 1. Make the necessary changes below to indicate that the class LargeAnimal is derived from the base class ZooAnimal. Specify that the base class is public.

```
class ZooAnimal
{
 private:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

class LargeAnimal
{
 private:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, float); // constructor function
```

```
    inline ~LargeAnimal () { delete [] species; }; // destructor function
    float reptCageMinimumVolume ();
};
```

Bài 2. On the cout statement for gonzo (last statement below), since the gonzo object is of type class LargeAnimal, gonzo's species will be returned by the LargeAnimal class' reptName member function. Make the necessary changes so that gonzo's name will be returned instead by the ZooAnimal class' reptName member function.

```
class ZooAnimal
{
 private:
    char* name;
    int cageNumber;
    int weightDate;
    int weight;
 public:
    ZooAnimal (char*, int, int, int); // constructor function
    inline ~ZooAnimal () { delete [] name; }; // destructor function
    void changeWeight (int pounds);
    char* reptName ();
    int reptWeight ();
    int daysSinceLastWeighed (int today);
};

// -------- member function to return the animal's name
char* ZooAnimal::reptName ()
{
 return name;
}

class LargeAnimal : public ZooAnimal
{
 private:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, char*, float); // constructor function
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
   char* reptName ();
};

// -------- member function to return the large animal's species
char* LargeAnimal::reptName ()
{
 return species;
}

// ========== an application to use the LargeAnimal class
void main ()
{
 ZooAnimal bozo;
 LargeAnimal gonzo;
 ...
 cout << bozo.reptName () << endl;
 cout << gonzo.reptName () << endl;
}
```

Bài 3. Make the necessary changes below so that all calls to the ZooAnimal member function reptName are passed to the matching function in the derived type when called for an object of the derived type.

```
class ZooAnimal
{
 private:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

// -------- member function to return the animal's name
char* ZooAnimal::reptName ()
{
 return name;
}

class LargeAnimal : public ZooAnimal
{
 private:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, char*, float); // constructor function
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
   char* reptName ();
};

// -------- member function to return the large animal's species
char* LargeAnimal::reptName ()
{
 return species;
}
```

Bài 4. Make the necessary changes to indicate that the class LargeAnimal is a derived class of the base classes ZooAnimal and Mammal (in that order). Specify that the base classes are public.

```
class ZooAnimal
{
 protected:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
```

```
    int daysSinceLastWeighed (int today);
  };

  class Mammal
  {
   protected:
     float minimumVolume;
     int minimumWeight;
   public:
     Mammal (float, int); // constructor function
     inline ~Mammal () {}; // destructor function
     float reptminimumVolume ();
     int reptminimumWeight ();
  };

  class LargeAnimal
  {
   protected:
     char* species;
     float cageMinimumVolume;
   public:
     LargeAnimal (char*, int, int, int, float, float, int); // constructor
     inline ~LargeAnimal () { delete [] species; }; // destructor function
     float reptCageMinimumVolume ();
  };
```

Bài 5. In the reptCageMinimumVolume function below, the reference to the data name weight is ambiguous. Make the necessary changes to indicate that the weight to be used is the one from the Mammal base class.

```
class ZooAnimal
{
 protected:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

class Mammal
{
 protected:
   float minimumVolume;
   int weight;
 public:
   Mammal (float, int); // constructor function
   inline ~Mammal () {}; // destructor function
   float reptminimumVolume ();
   int reptWeight ();
};

class LargeAnimal : public ZooAnimal, public Mammal
{
 protected:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, float, float, int); // constructor
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
};

// -------- member function to return the minimum cage volume
// -------- needed for this large animal
float LargeAnimal::reptCageMinimumVolume ()
{
 if (weight < 500)
    return cageMinimumVolume;
 else
    return reptminimumVolume ();
}
```

Bài 6. In the reptCageMinimumVolume function below, the reference to the function name reptWeight is ambiguous. Make the necessary changes to indicate that the reptWeight function to be used is the one from the ZooAnimal base class.

```
class ZooAnimal
{
 protected:
   char* name;
   int cageNumber;
   int weightDate;
   int weight;
 public:
   ZooAnimal (char*, int, int, int); // constructor function
   inline ~ZooAnimal () { delete [] name; }; // destructor function
   void changeWeight (int pounds);
   char* reptName ();
   int reptWeight ();
   int daysSinceLastWeighed (int today);
};

class Mammal
{
 protected:
   float minimumVolume;
   int weight;
 public:
   Mammal (float, int); // constructor function
   inline ~Mammal () {}; // destructor function
   float reptminimumVolume ();
   int reptWeight ();
};

class LargeAnimal : public ZooAnimal, public Mammal
{
 protected:
   char* species;
   float cageMinimumVolume;
 public:
   LargeAnimal (char*, int, int, int, float, float, int); // constructor
   inline ~LargeAnimal () { delete [] species; }; // destructor function
   float reptCageMinimumVolume ();
};
```

```
  // -------- member function to return the minimum cage volume
  // -------- needed for this large animal
  float LargeAnimal::reptCageMinimumVolume ()
  {
   if (Mammal::weight < 500)
      return cageMinimumVolume;
   else
      return reptWeight ();
  }
```

Bài 7. What do you think this program will output?

```
class A
{
public:
   virtual const char* getName() { return "A"; }
};

class B: public A
{
public:
   virtual const char* getName() { return "B"; }
};

class C: public B
{
public:
   virtual const char* getName() { return "C"; }
};

class D: public C
```

```
{
public:
    virtual const char* getName() { return "D"; }
};

int main()
{
    C c;
    A &rBase = c;
    std::cout << "rBase is a " << rBase.getName() << '\n';

    return 0;
}
```

Bài 8. weite a program for below requirement.

class dev1:

- account:
- level: string level1.1 to level1.9
- coin: only have 50%, 50% project Manager will keep.

class dev2:

- account:
- level: string level2.1 to level2.9
- coin: 100% coin

class projectManager:

- account:
- level: string pm1 to pm4
- coin: 25% coin of all devs's total coins.

Note: coin is counted by milion (4500000 VND = 4.5) we have clase employee

```
class employee
{
protected:
   std::string m_name;

   // We're making this constructor protected because
   // we don't want people creating employee objects directly,
   // but we still want inhertance classes to be able to use it.
   employee(std::string name)
      : m_name(name)
   {
   }

public:
   std::string getName() { return m_name; }
   const char* level() { return "level0.0"; }
   double coin(){return 0.0;}
};
```

class dev1, dev2, projectManager will inheritance from class employee.

They will re-write 2 functions level() and coin().

We have:

dev1:

- AnLN: level1.3, coin 7.5
- BinhTT2: level1.5, coin 6.3
- ChungCH: level1.3, coin 9.0
- DucNV2: level1.8, coin 5.8

dev2:

- HaLN: level2.3, coin 10.5
- LinhTT2: level2.5, coin 9.3
- HinhTT2: level2.9, coin 12.3

projectManager:

- MinhTT2: pm3

Print all the devs and projectManager with name, level and coins.