Exercise 4: Displaying Message Using Virtual Functions

This exercise requires you to implement the concept of virtual functions to display information about two base classes.

We'll cover the following

Problem Statement

You will first build **three** *classes*:

- `Mammal` (*parent class*)
- `Dog` (*derived class*)
- `Cat` (*derived class*)

`Dog` and `Cat` class will inherit from `Mammal`.

In the exercise you have to implement the following:

- `Mammal` class:
  - Has one `protected` variable for **age** of the mammal.
  - A constructor that takes the age of mammal as input and sets it.
  - The function `Eat()` that displays "Mammal eats food".
  - Function `Speak()` that displays "Mammal speaks mammalian!!".
  - Function `get_Age()` which returns the age of the mammal.
- `Dog` class:
  - Inherits all the *members* from `Mammal` class.
  - Implement all `member` functions of `Mammal` class for `Dog` class.
  - `Eat()` should display **"Dog eats meat"**.
  - `Speak()` should display **"Dog barks: ruff! ruff!"**.
  - `get_Age()` should *return* Dog's **age**.

- `Cat` class:
    - Inherits all the *members* from `Mammal` class.
    - Implement all *member* functions of `Mammal` class for `Cat` class.
    - `Eat()` should display **“Cat eats meat”**.
    - `Speak()` should display **“Cat meows: Meow! Meow!”**.
    - `get_Age()` should *return* Cat’s **age**.

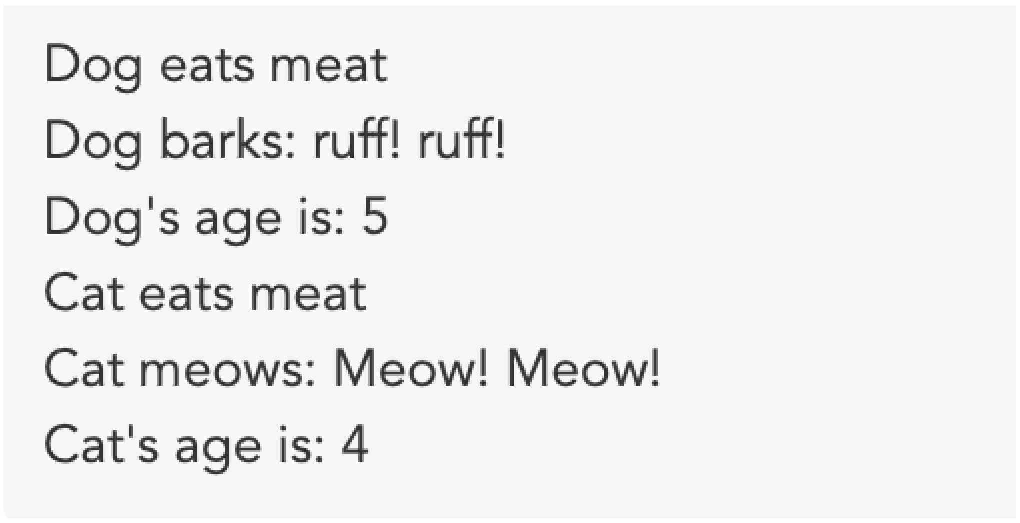
> **Hint**: Think along the direction of `virtual` functions and their use to implement the **same** *function* for **different** *classes* separately.

Here’s a sample result which you should get.

Input:

```
Dog d(5);
Cat c(4);
```

Expected Output:



```
Dog eats meat
Dog barks: ruff! ruff!
Dog's age is: 5
Cat eats meat
Cat meows: Meow! Meow!
Cat's age is: 4
```

Expected Output

Write your code below. It is recommended that you try solving the exercise yourself before viewing the solution.

Good Luck!

Exercise | Solution

```
#include <iostream>
using namespace std;


class Mammal
{
public:
    Mammal(int age){
      itsAge=age;
}
    virtual void Eat() {cout << "Mammal eats food"<<endl;}
    virtual void Speak() {cout << "Mammal speaks mammalian!!"<<endl;}
    virtual int get_Age(){return itsAge;}

protected:
    int itsAge;
};

class Dog: public Mammal{
  public:
         Dog(int age=0) : Mammal(age) {}
         void Eat() {cout << "Dog eats meat"<<endl;}
         void Speak() {cout << "Dog barks: ruff! ruff!"<<endl;}
         int get_Age(){return itsAge;}
};

class Cat: public Mammal{
  public:
         Cat(int age=0): Mammal(age){}
         void Eat() {cout << "Cat eats meat"<<endl;}
         void Speak() {cout << "Cat meows: Meow! Meow!" <<endl;}
         int get_Age(){return itsAge;}
};

int main(void) {
   Mammal *m;
   Dog doggo(5); //making object of child class Dog
   Cat catty(4); //making object of child class Cat

   m = &doggo;
   m->Eat();
   m->Speak();
   cout << "Dog's age is: "<<m->get_Age()<<endl;
   m= &catty;
   m->Eat();
   m->Speak();
   cout << "Cat's age is: "<<m->get_Age()<<endl;


   return 0;
}
```