# C++ Chapter 7: Inheritance

## Contents

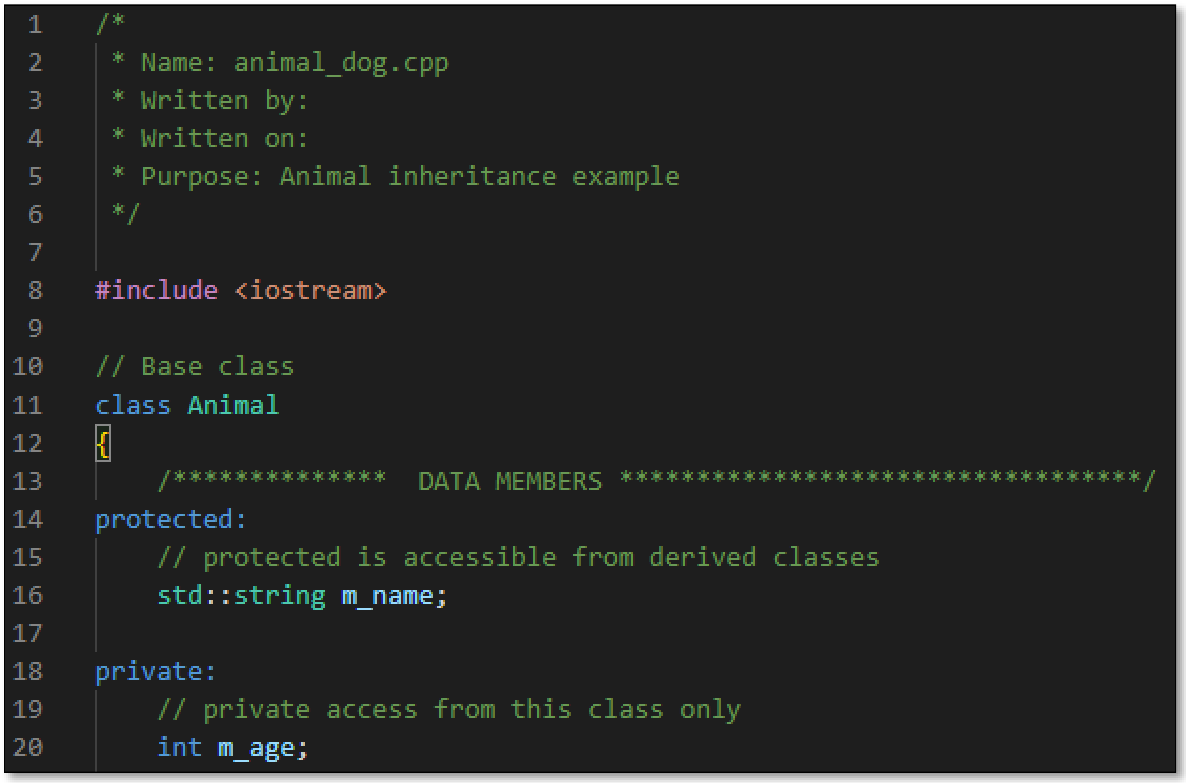
Time required: 90 minutes

## Inheritance

Inheritance is a fundamental concept in object-oriented programming (OOP) and is supported in the C++ programming language. It allows a class to inherit the properties (data members) and behavior (member functions) of another class, known as the base or parent class. The class that inherits from the base class is called the derived or child class. Inheritance allows for code reuse and facilitates the creation of class hierarchies.

## Tutorial 1: Animal Inheritance

**Defining a Base Class:** A base class is defined using the **class** keyword followed by the class name. It can have data members (variables) and member functions (methods).

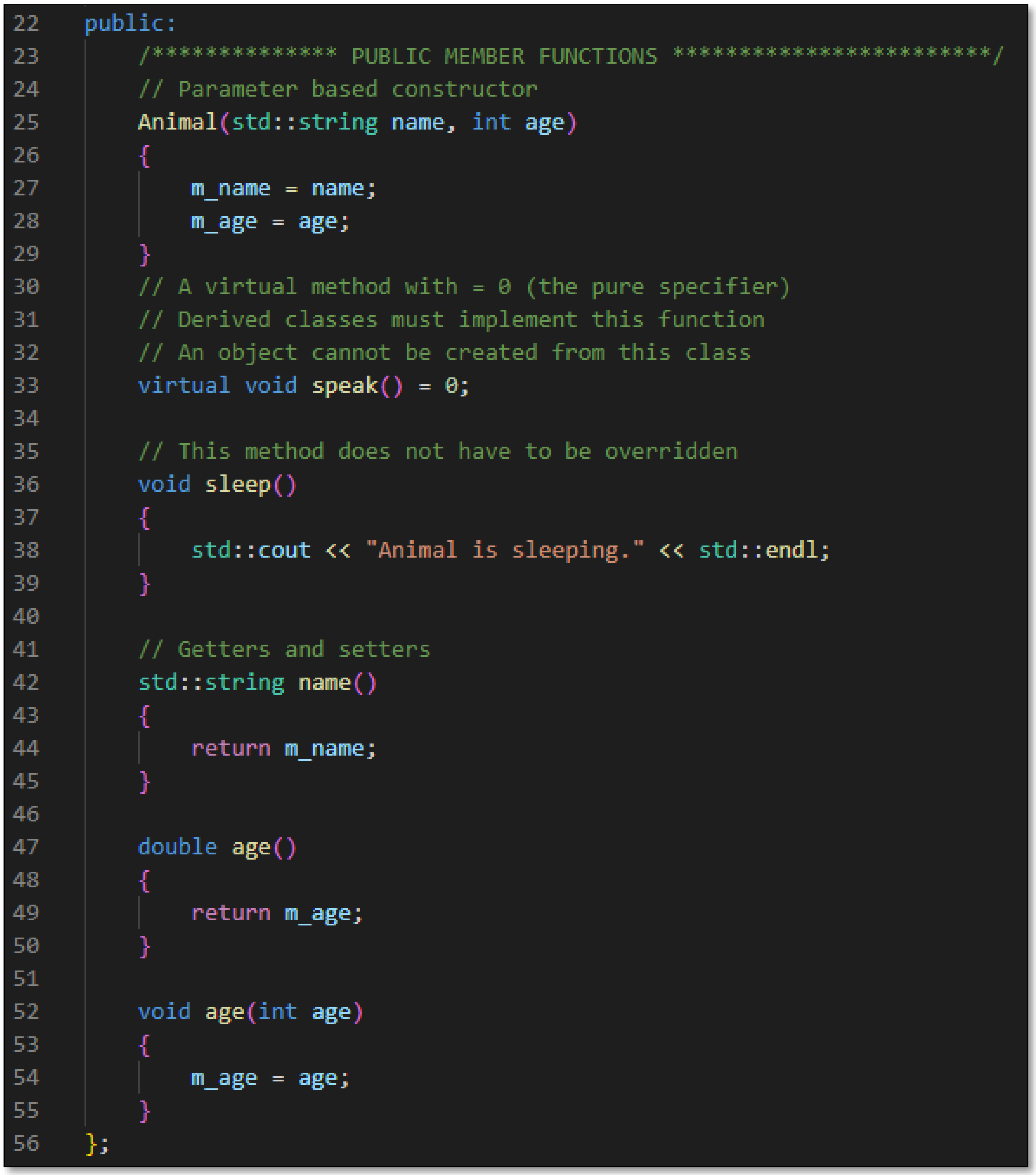


```cpp
/*
 * Name: animal_dog.cpp
 * Written by:
 * Written on:
 * Purpose: Animal inheritance example
 */

#include <iostream>

// Base class
class Animal
{
    /**************  DATA MEMBERS ***********************************/
protected:
    // protected is accessible from derived classes
    std::string m_name;

private:
    // private access from this class only
    int m_age;
```



```cpp
public:
    /************** PUBLIC MEMBER FUNCTIONS ************************/
    // Parameter based constructor
    Animal(std::string name, int age)
    {
        m_name = name;
        m_age = age;
    }
    // A virtual method with = 0 (the pure specifier)
    // Derived classes must implement this function
    // An object cannot be created from this class
    virtual void speak() = 0;

    // This method does not have to be overridden
    void sleep()
    {
        std::cout << "Animal is sleeping." << std::endl;
    }

    // Getters and setters
    std::string name()
    {
        return m_name;
    }

    double age()
    {
        return m_age;
    }

    void age(int age)
    {
        m_age = age;
    }
};
```

## Deriving a Class

Derived classes are classes that inherit properties and behavior from a base class. To create a derived class, you use the colon `:` followed by the access specifier (`public`, `private`, or `protected`) and the name of the base class.

```
// DerivedClass inherits from the BaseClass
```

```
class DerivedClass : public BaseClass
```

## Overriding Member Functions

Child classes can override the member functions of the base class to provide their own implementation. To do this, you define a member function with the same name, return type, and parameters in the derived class.

## Access Specifiers

The access specifiers (`public`, `private`, and `protected`) determine the visibility and accessibility of the base class members in the derived class. In C++, the default access specifier for class members is `private` if not specified otherwise.

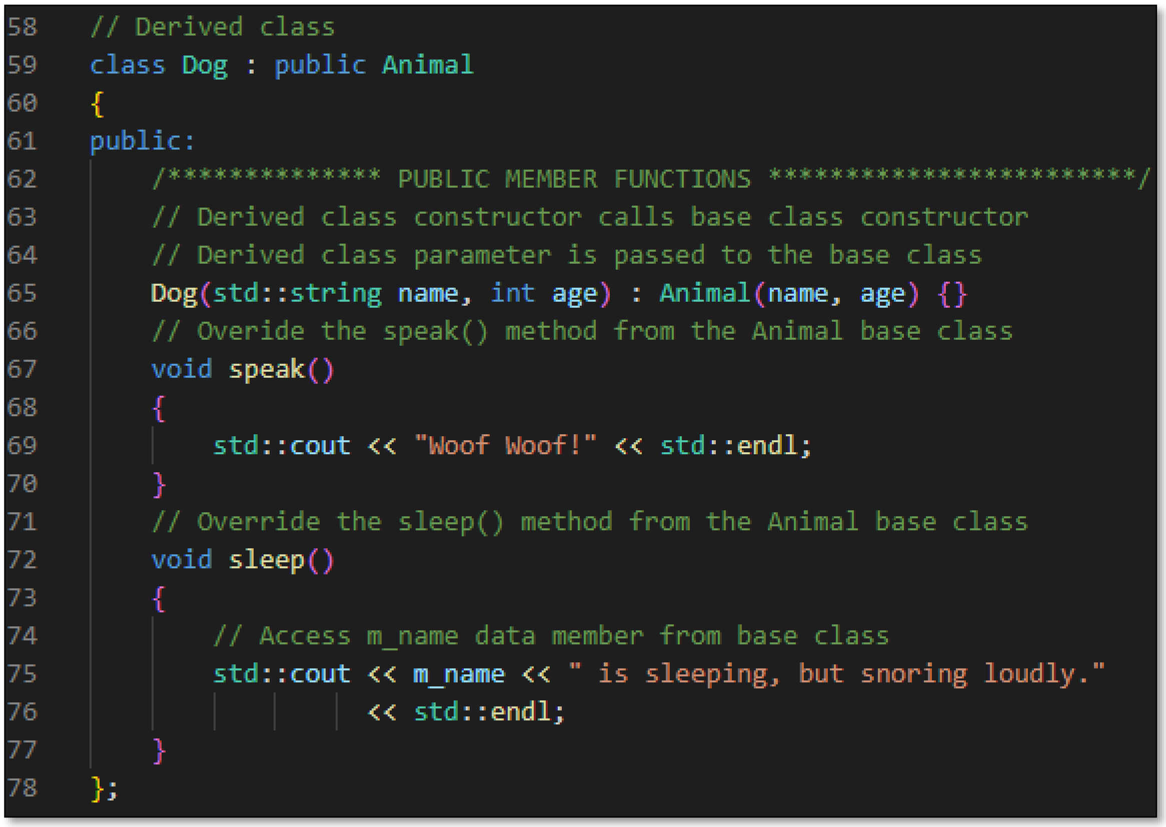
- **public:** Public members of the base class are accessible in the derived class as if they were defined in the derived class itself.
- **private:** Private members of the base class are not accessible in the derived class.
- **protected:** Protected members of the base class are accessible in the derived class, but not accessible outside the class hierarchy.

## Pure Virtual Functions and Abstract Classes

In C++, "pure virtual" functions, also known as "abstract" functions, are declared and do not have an implementation in the class where they are declared. Derived classes that inherit from an abstract class with pure virtual functions must provide their own implementation for those functions.

When it comes to inheritance with pure virtual functions in C++, the derived class must implement all the pure virtual functions inherited from its base classes to become a concrete class that can be instantiated. Failure to provide an implementation for all pure virtual functions will result in the derived class also being treated as abstract, and objects of that class cannot be created.

Animal is an abstract class. It cannot be used to create an object.



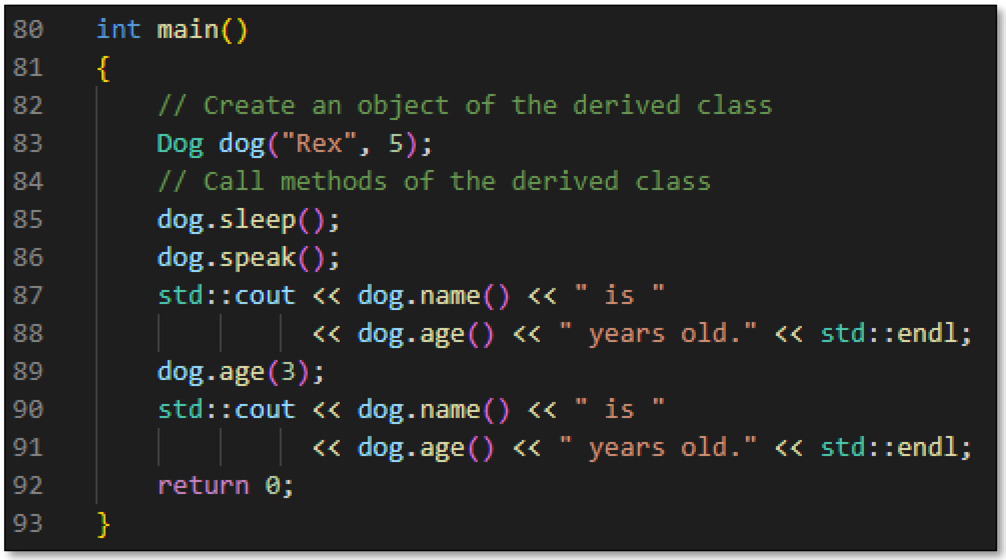
```cpp
// Derived class
class Dog : public Animal
{
public:
    /************** PUBLIC MEMBER FUNCTIONS ************************/
    // Derived class constructor calls base class constructor
    // Derived class parameter is passed to the base class
    Dog(std::string name, int age) : Animal(name, age) {}
    // Overide the speak() method from the Animal base class
    void speak()
    {
        std::cout << "Woof Woof!" << std::endl;
    }
    // Override the sleep() method from the Animal base class
    void sleep()
    {
        // Access m_name data member from base class
        std::cout << m_name << " is sleeping, but snoring loudly."
                  << std::endl;
    }
};
```

In this example, the **Dog** class is derived from the **Animal** class with **public** inheritance. This means that the public members of the **Animal** class are accessible in the **Dog** class.

The **speak()** function in the Dog class overrides the **speak()** function in the **Animal** class. When you call **speak()** on a Dog object, the overridden implementation in the **Dog** class will be invoked.

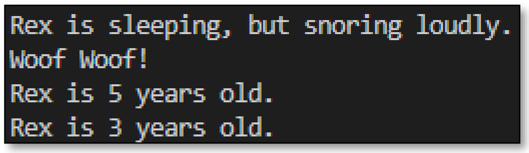
## Creating Objects

You create objects of the derived class using its constructor. The base class is abstract, you can't create an object from this class because of the pure virtual method.



```cpp
int main()
{
    // Create an object of the derived class
    Dog dog("Rex", 5);
    // Call methods of the derived class
    dog.sleep();
    dog.speak();
    std::cout << dog.name() << " is "
              << dog.age() << " years old." << std::endl;
    dog.age(3);
    std::cout << dog.name() << " is "
              << dog.age() << " years old." << std::endl;
    return 0;
}
```

Example run:



```
Rex is sleeping, but snoring loudly.
Woof Woof!
Rex is 5 years old.
Rex is 3 years old.
```

## Assignment 1: Cats and Dogs

Time to practice. Add the following to the tutorial program.

### Animal Class

- Add another data member
- Add a getter and setter

### Another Animal Class

You can choose your own animal. It can real or fictional.

- Create another animal which derives from the Animal class
- Override the **speak()** and **sleep()** methods

Create appropriate objects to demonstrate the new data member and the new class.

## Assignment Submission

1. Attach the program files.

2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.