Designing Good Classes / Compiling C++ Programs

CSci-3081W: Program Design and Development
Professor Daniel Keefe

Continuing from last time…

Don’t Expose Member Data in Public

- A Point class needs x,y data, which is better?

```
class Point {

public:
  float x;
  float y;
};
```

```
class Point {

public:
  float GetX();
  float GetY();

  void SetX( float x );
  void SetY( float y );

private:
  float x;
  float y;
};
```

Can you design a good C++ class interface now?

- We just looked at a 2D point.
- Now, how about an ADT for a Circle? You design it.

  - I’d like to be able to adjust its radius as my program executes.
  - I’d also like to be able to calculate the area of the circle.
  - And, I might want to move the center point (x,y) of the circle.

The next slides contain the most important concept for designing good object oriented programs!

Good Class Design: Two Types of Relationships between Classes: "has a" and "is a"

(note the triple underline,
the professor must think this is important)

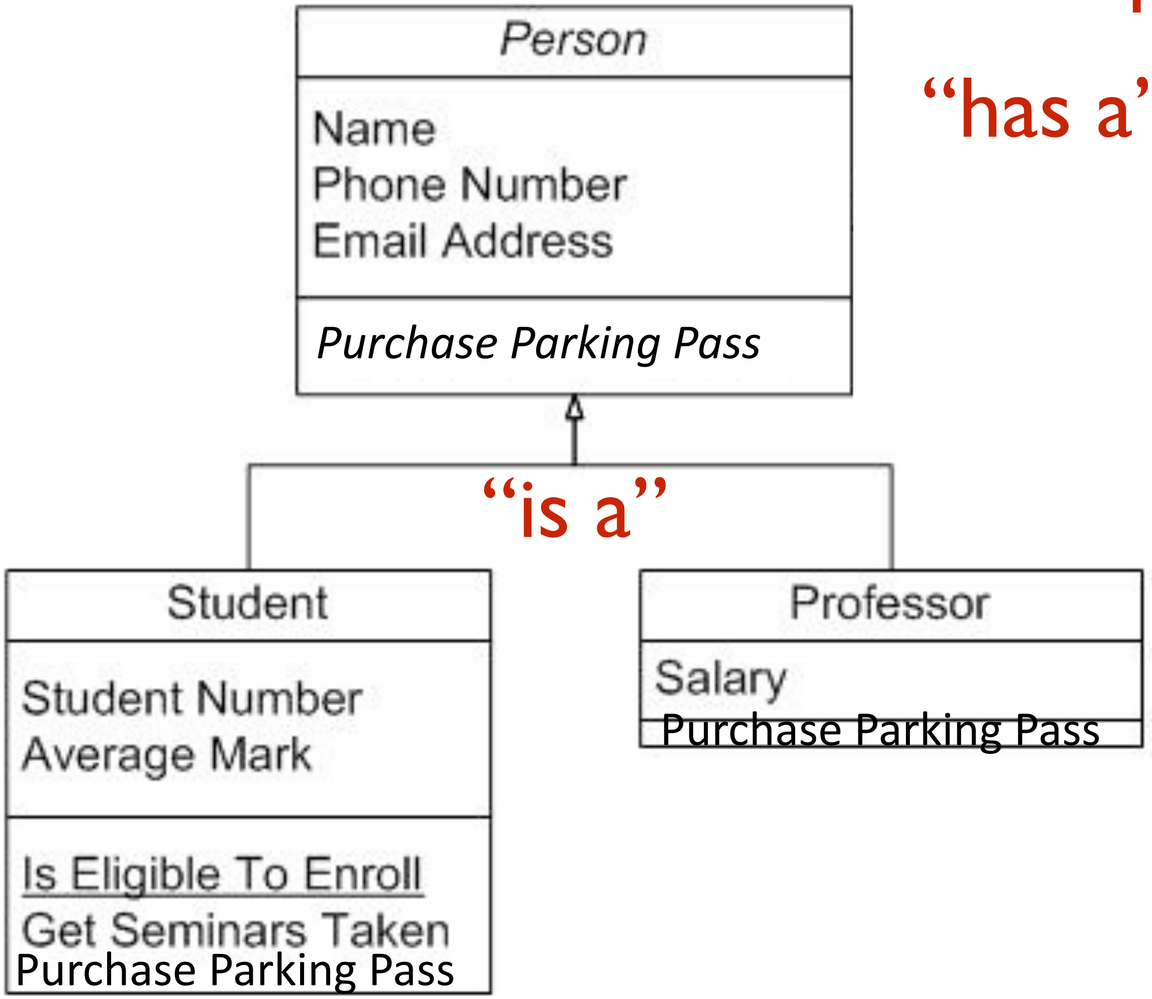
C++ Example

```
class Shape {
public:
  // virtual means subclasses can override this function
  virtual float area();
}

class Square : public Shape {
public:
  float area();
}
```

Another Example (by the way this is a UML diagram)

Example of Containment

“has a”



Person

Name
Phone Number
Email Address

Purchase Parking Pass

“is a”

Student

Student Number
Average Mark

Is Eligible To Enroll
Get Seminars Taken
Purchase Parking Pass

Professor

Salary
Purchase Parking Pass

Example of Inheritance

Today, Part 1: Designing Good Classes

Two Types of Relationships between Classes:

1. Inheritance (also known as “is a”)
2. Containment (also know as “has a”)

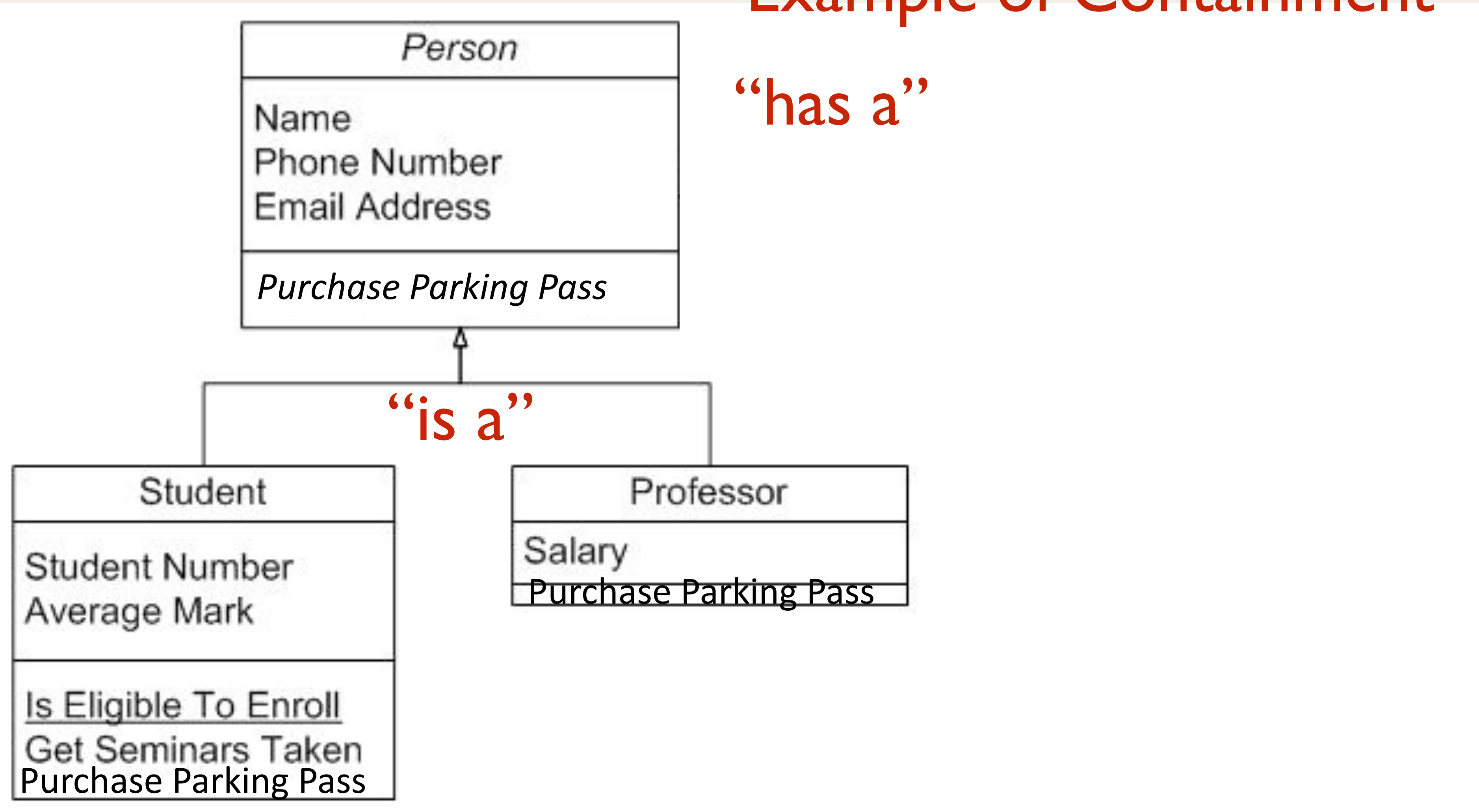
Example of an “is a” relationship implemented in C++

```
class Shape {
public:
  // virtual means subclasses can override this function
  virtual float area() = 0;
}


class Circle : public Shape {
public:
  float area();
}


class Square : public Shape {
public:
  float area();
}
```

important syntax

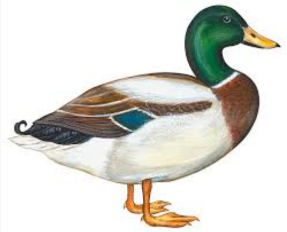
Another Example (by the way this is a UML diagram)

Example of Containment



Example of Inheritance

You try an example: Ducks

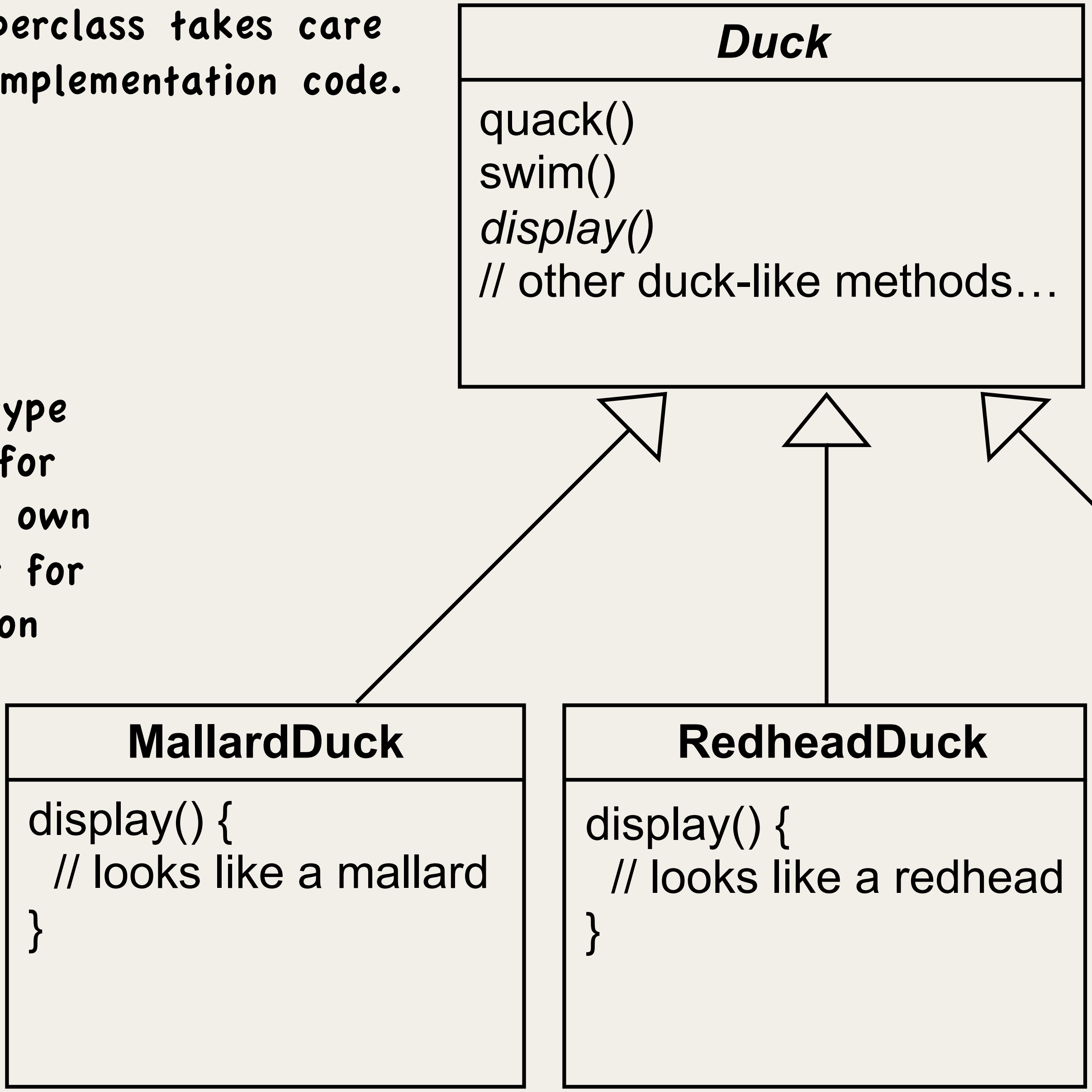


Mallard ,

RedHead , . . .

- All ducks **quack()** and **swim()**, the superclass takes care of the implementation details.
- Each duck subtype is responsible for implementing its own **display()** method, since they all look different when drawn on the screen.

All ducks quack and swim, the superclass takes care of the implementation code.



Duck

```
quack()
swim()
display()
// other duck-like methods…
```

MallardDuck

```
display() {
  // looks like a mallard
}
```

RedheadDuck

```
display() {
  // looks like a redhead
}
```

The display() method is abstract, since all duck subtypes look different

Each duck subtype is responsible for implementing its own display behavior for how it looks on the screen.

Lots of other types of ducks inherit from the Duck class

Inheritance vs. Containment

- These are the two most common relationships that are used when designing programs based on Abstract Data Types (ADTs).
- It's not always obvious which one to use. *Next week, we're going to look at a very interesting example of this!*
- A common mistake is to use too much Inheritance, forcing classes into an "is a" relationship when it doesn't quite fit.
- With practice, you should learn to identify this situation and avoid it — we'll come back to this throughout the semester.

Part 2: Building C++ Programs

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

What you need to learn about building C++ programs

- There are two steps to building a C++ program:

  1. compiling
  2. linking

- Compiling: Every .cpp file that is part of the program needs to be compiled separately into an object file.
  - File1.cpp gets compiled into File1.o
  - File2.cpp gets compiled into File2.o
  - …

- Linking: All of the resulting .o files are then linked together to create an executable program.
  - File1.o, File2.o, and … all get linked together to create MyProgram.exe (or just MyProgram on linux systems)

What you need to learn about building C++ programs

- On linux systems, the C++ compiler is a program called g++. You can run this directly from the command line:
    - g++ -c -o File1.o File1.cpp
    - g++ -c -o File2.o File2.cpp
    - g++ -c -o FileX.o FileX.cpp
- g++ can take many, many command line options. The most common is the -o flag, which specifies a name for the output file.
- Conveniently (but this can also be a bit confusing), the g++ program is also used to do the linking step. g++ automatically figures out whether you want it to compile or to link based on whether you pass it .cpp files or .o files.
    - g++ -o MyProgram File1.o File2.o FileX.o

What you need to learn about building C++ programs

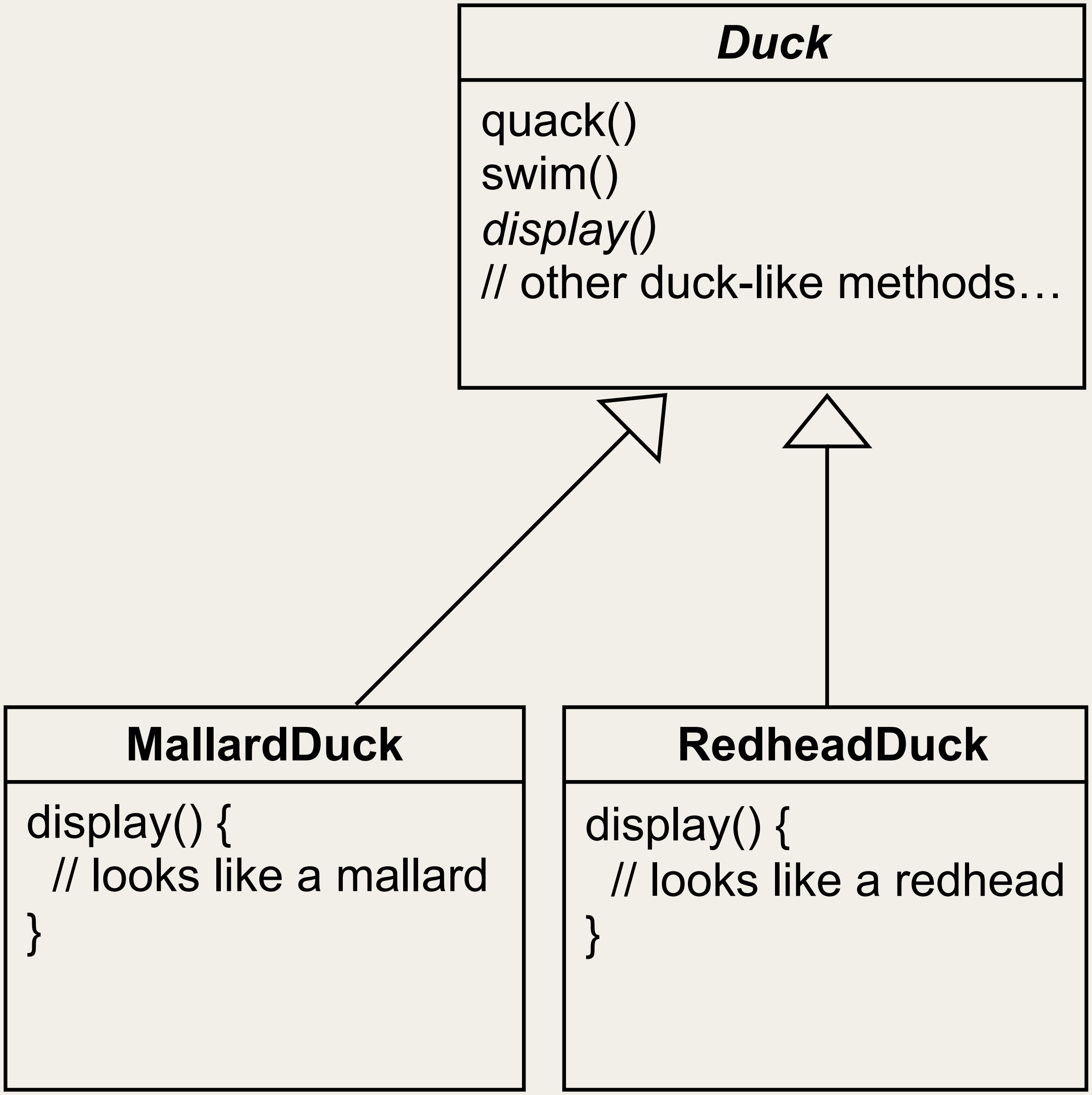
- What about the header (.h) files? Do they get compiled too?

- Yes, but only when a .cpp file needs them.

- For example in File1.cpp one of the first lines will likely be:

  ```
  #include “File1.h”
  ```

- So, when File1.cpp is read by the compiler and the compiler sees this #include directive, it includes the entire contents of File1.h in the compilation.



Duck

quack()
swim()
display()
// other duck-like methods…

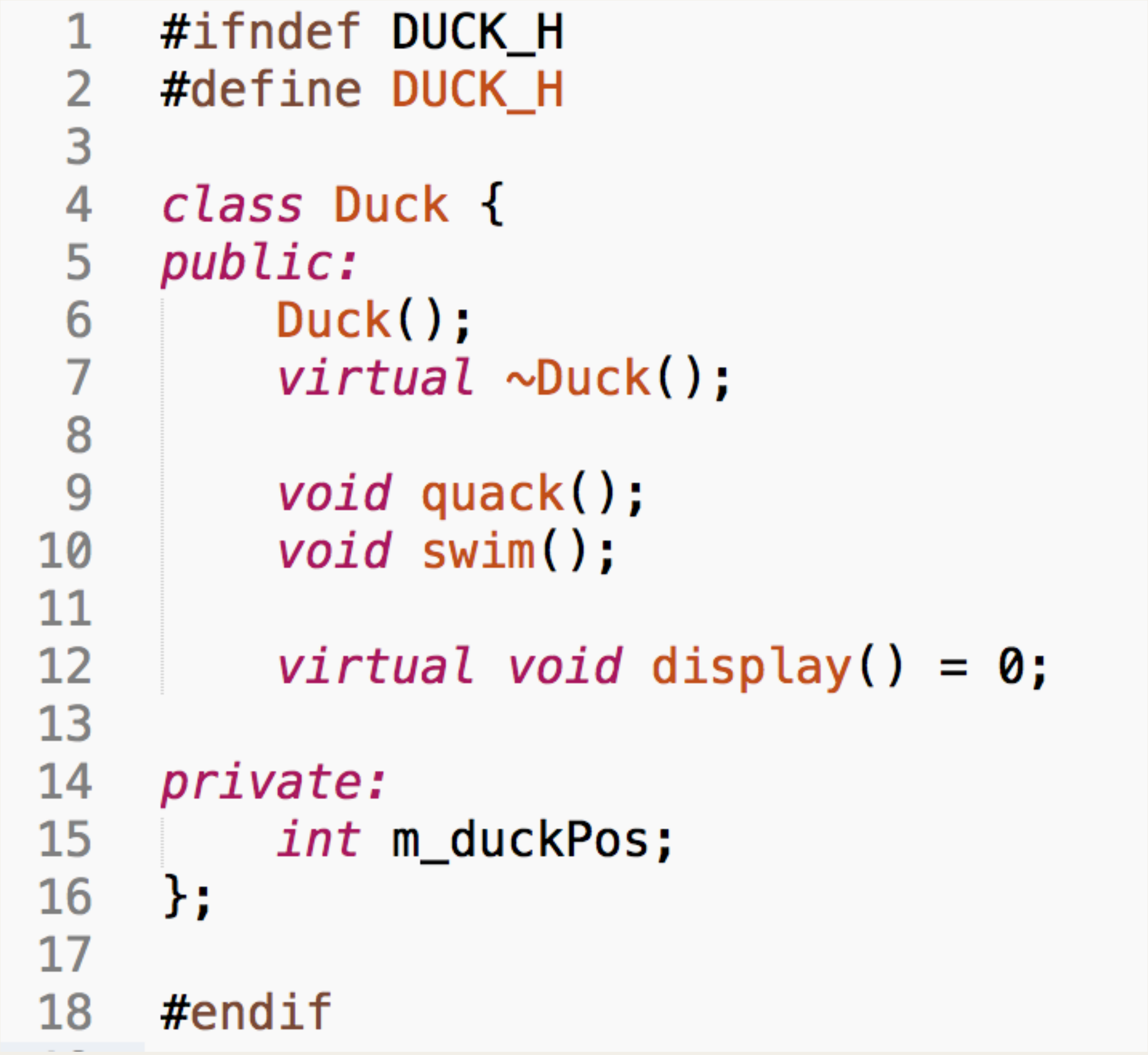
MallardDuck

```
display() {
  // looks like a mallard
}
```

RedheadDuck

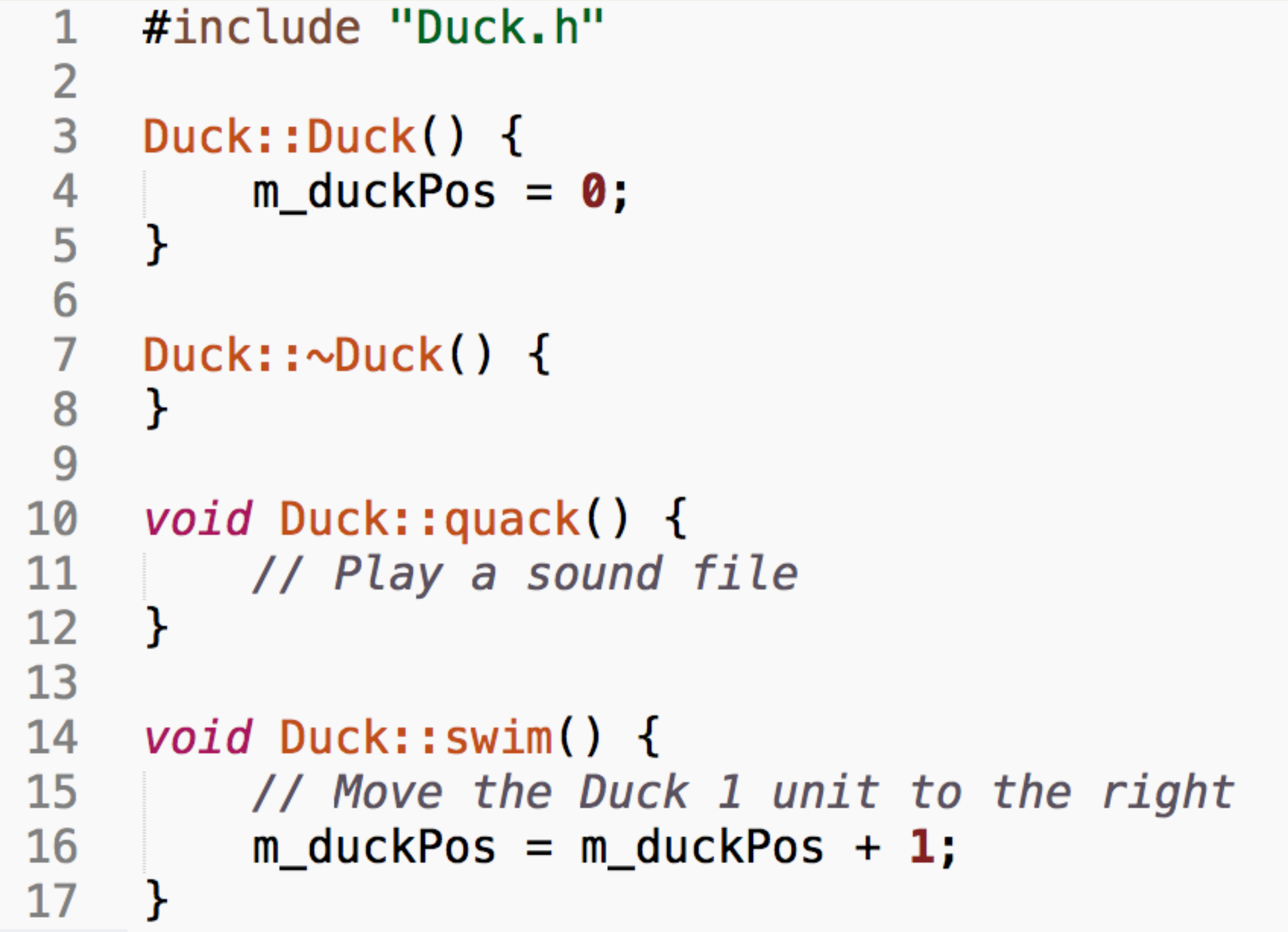
```
display() {
  // looks like a redhead
}
```

Duck.h



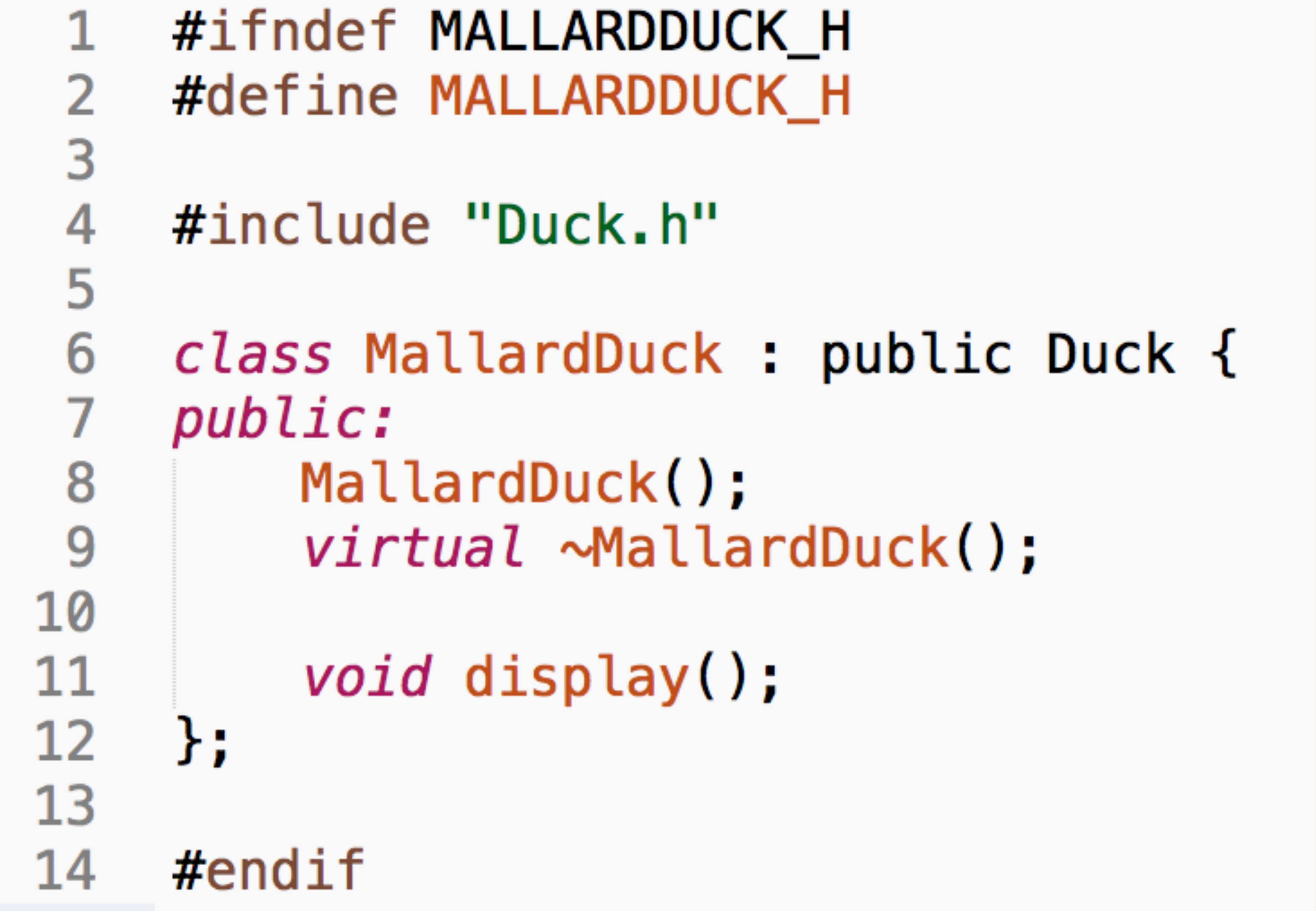
```
#ifndef DUCK_H
#define DUCK_H

class Duck {
public:
    Duck();
    virtual ~Duck();

    void quack();
    void swim();

    virtual void display() = 0;

private:
    int m_duckPos;
};

#endif
```

Duck.cpp



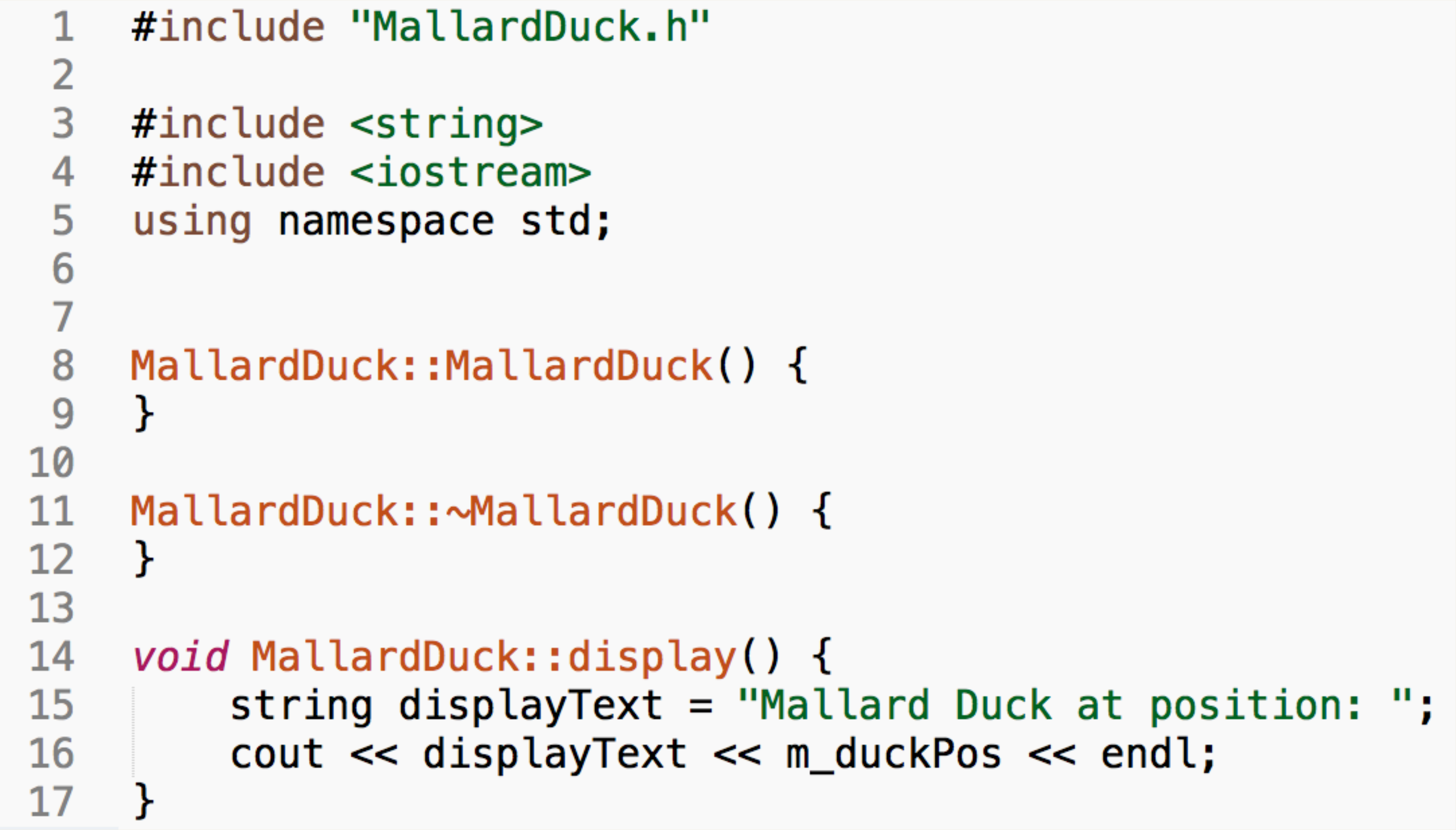
```cpp
#include "Duck.h"

Duck::Duck() {
    m_duckPos = 0;
}

Duck::~Duck() {
}

void Duck::quack() {
    // Play a sound file
}

void Duck::swim() {
    // Move the Duck 1 unit to the right
    m_duckPos = m_duckPos + 1;
}
```

MallardDuck.h



```cpp
#ifndef MALLARDDUCK_H
#define MALLARDDUCK_H

#include "Duck.h"

class MallardDuck : public Duck {
public:
    MallardDuck();
    virtual ~MallardDuck();

    void display();
};

#endif
```

MallardDuck.cpp

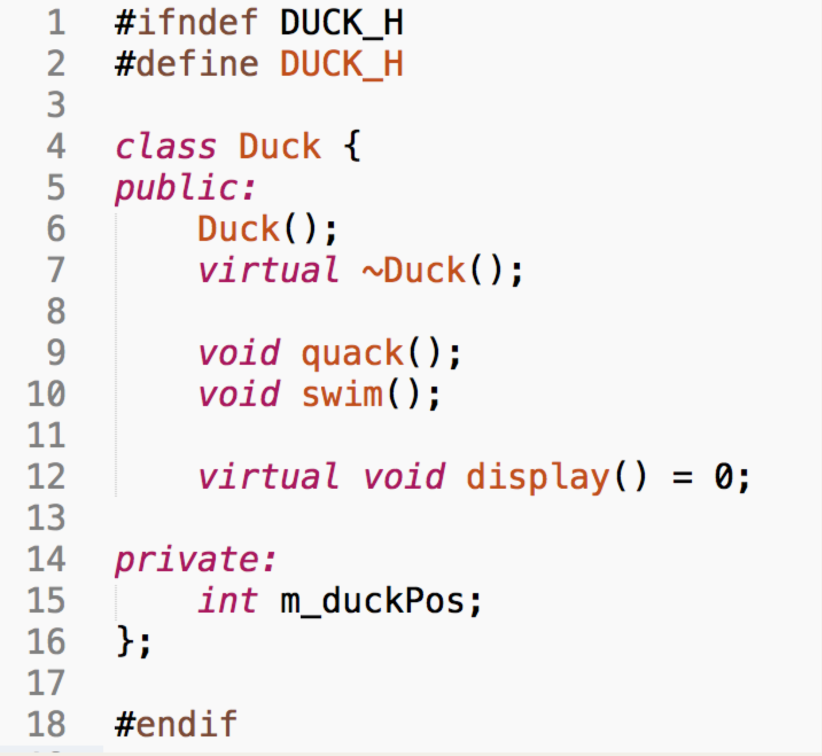


```cpp
#include "MallardDuck.h"

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


MallardDuck::MallardDuck() {
}

MallardDuck::~MallardDuck() {
}

void MallardDuck::display() {
    string displayText = "Mallard Duck at position: ";
    cout << displayText << m_duckPos << endl;
}
```

Ok, let’s compile this program!!!

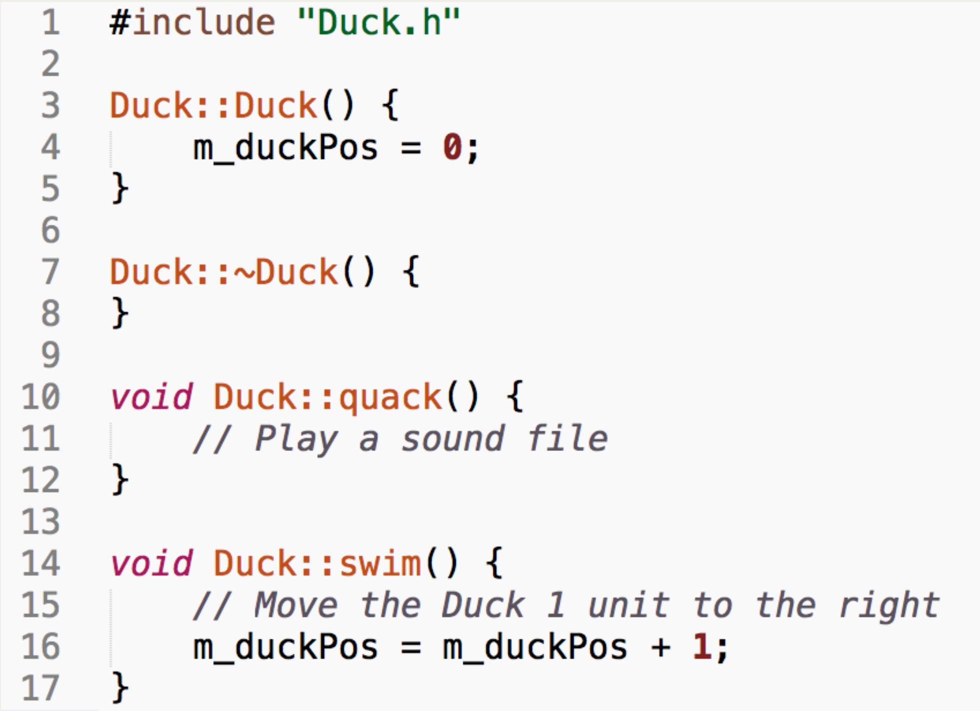
…by acting it out :)

Duck.h



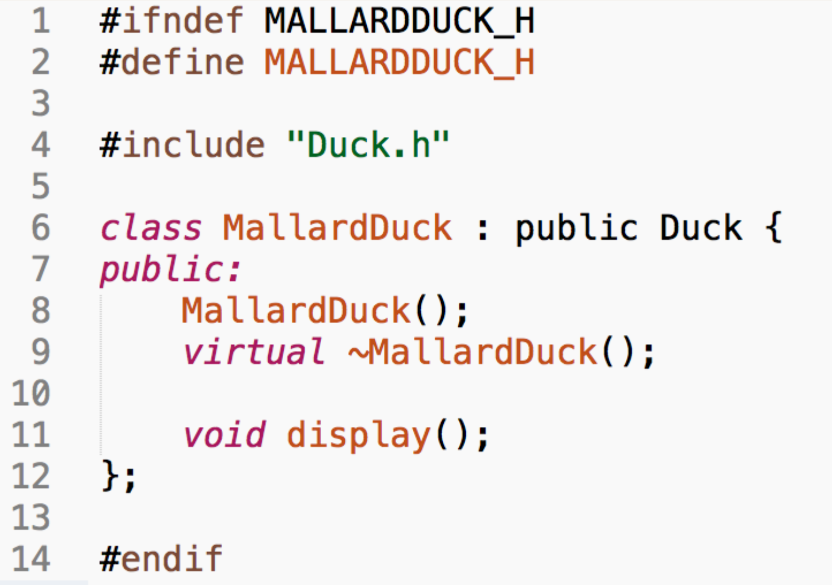
```cpp
#ifndef DUCK_H
#define DUCK_H

class Duck {
public:
    Duck();
    virtual ~Duck();

    void quack();
    void swim();

    virtual void display() = 0;

private:
    int m_duckPos;
};

#endif
```

Duck.cpp



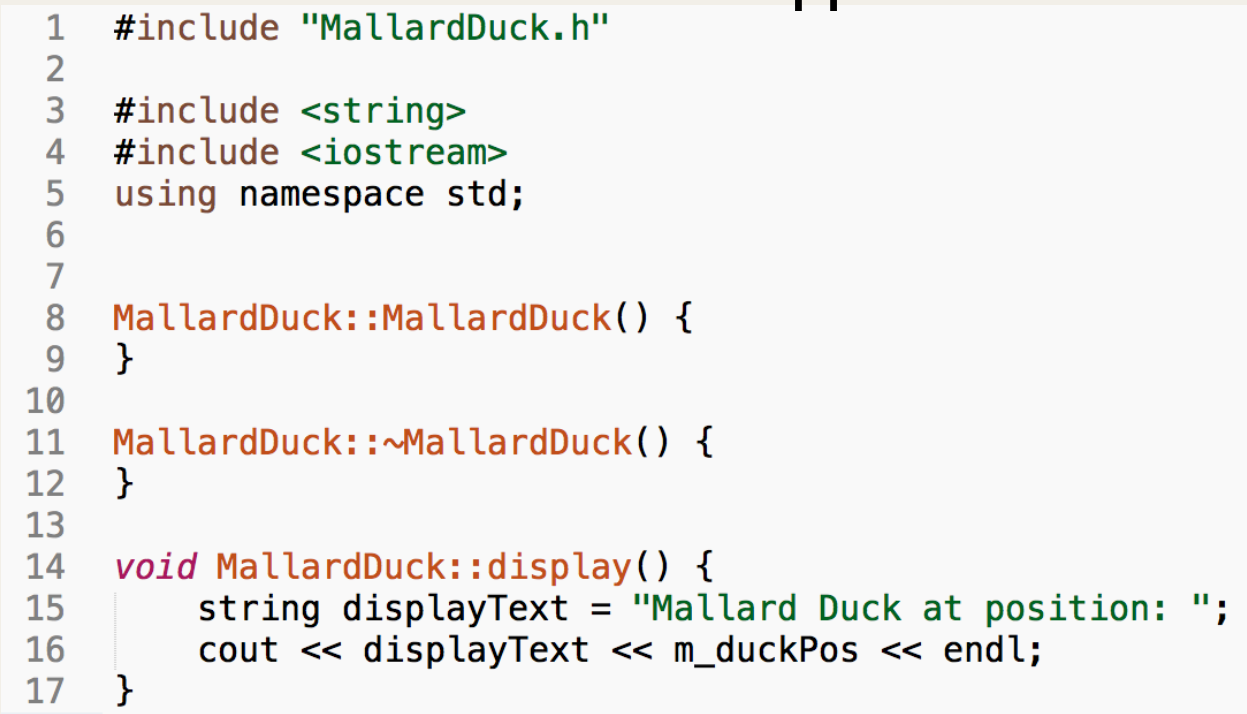
```cpp
#include "Duck.h"

Duck::Duck() {
    m_duckPos = 0;
}

Duck::~Duck() {
}

void Duck::quack() {
    // Play a sound file
}

void Duck::swim() {
    // Move the Duck 1 unit to the right
    m_duckPos = m_duckPos + 1;
}
```

MallardDuck.h



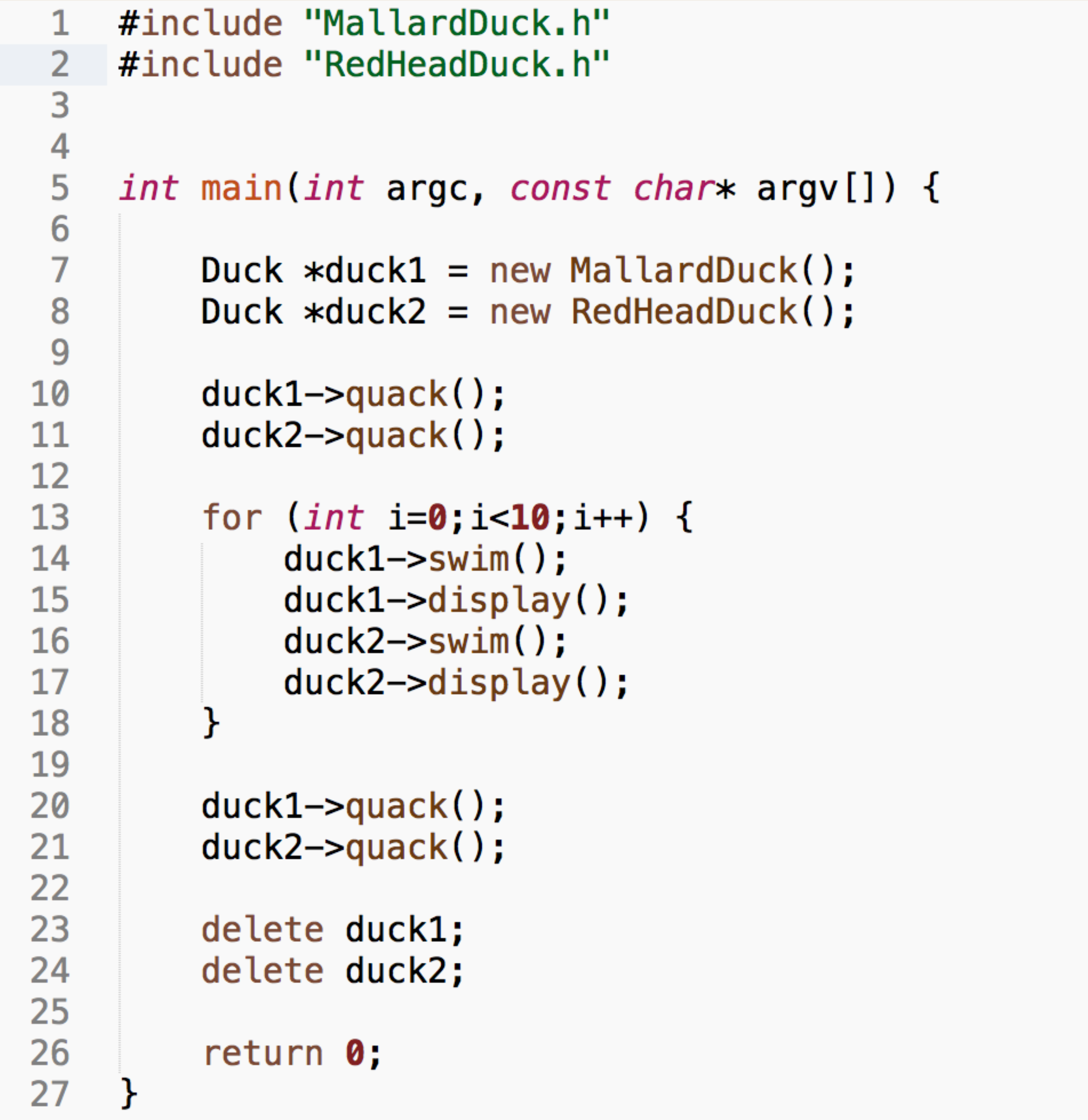
```cpp
#ifndef MALLARDDUCK_H
#define MALLARDDUCK_H

#include "Duck.h"

class MallardDuck : public Duck {
public:
    MallardDuck();
    virtual ~MallardDuck();

    void display();
};

#endif
```

MallardDuck.cpp



```cpp
#include "MallardDuck.h"

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


MallardDuck::MallardDuck() {
}

MallardDuck::~MallardDuck() {
}

void MallardDuck::display() {
    string displayText = "Mallard Duck at position: ";
    cout << displayText << m_duckPos << endl;
}
```

main.cpp



```cpp
#include "MallardDuck.h"
#include "RedHeadDuck.h"


int main(int argc, const char* argv[]) {

    Duck *duck1 = new MallardDuck();
    Duck *duck2 = new RedHeadDuck();

    duck1->quack();
    duck2->quack();

    for (int i=0;i<10;i++) {
        duck1->swim();
        duck1->display();
        duck2->swim();
        duck2->display();
    }

    duck1->quack();
    duck2->quack();

    delete duck1;
    delete duck2;

    return 0;
}
```

Duck.h

Duck.cpp

Duck.o

MallardDuck.h

MallardDuck.cpp

MallardDuck.o

The compiler:
g++

<string>

refers to a standard C++ include file: /usr/include/c++/4.4.3/string

<iostream>

refers to a standard C++ include file:

/usr/include/c++/4.4.3/iostream

main.cpp

main.o

MyDuckProgram

The Programmer