8.15 — Nested types in classes

BY ALEX ON DECEMBER 21ST, 2016 | LAST MODIFIED BY ALEX ON JANUARY 4TH, 2018

Consider the following short program:

```
#include <iostream>

enum FruitType
{
    APPLE,
    BANANA,
    CHERRY
};

class Fruit
{
private:
    FruitType m_type;
    int m_percentageEaten = 0;

public:


    Fruit(FruitType type) :
        m_type(type)
    {
    }

    FruitType getType() { return m_type;  }
    int getPercentageEaten() { return m_percentageEaten;  }
};

int main()
{
    Fruit apple(APPLE);

    if (apple.getType() == APPLE)
        std::cout << "I am an apple";
    else
        std::cout << "I am not an apple";

    return 0;
}
```

There's nothing wrong with this program. But because enum FruitType is meant to be used in conjunction with the Fruit class, it's a little weird to have it exist independently from the class itself.

Nesting types

Unlike functions, which can't be nested inside each other, in C++, types can be defined (nested) inside of a class. To do this, you simply define the type inside the class, under the appropriate access specifier.

Here's the same program as above, with FruitType defined inside the class:

```
#include <iostream>

class Fruit
{
public:
    // Note: we've moved FruitType inside the class, under the public access specifier
    enum FruitType
    {
        APPLE,
```

```
        BANANA,
        CHERRY
    };

private:
    FruitType m_type;
    int m_percentageEaten = 0;

public:


    Fruit(FruitType type) :
        m_type(type)
    {
    }

    FruitType getType() { return m_type;  }
    int getPercentageEaten() { return m_percentageEaten;  }
};

int main()
{
    // Note: we access the FruitType via Fruit now
    Fruit apple(Fruit::APPLE);

    if (apple.getType() == Fruit::APPLE)
        std::cout << "I am an apple";
    else
        std::cout << "I am not an apple";

    return 0;
}
```

First, note that FruitType is now defined inside the class. Second, note that we've defined it under the public access specifier, so the type definition can be accessed from outside the class.

Classes essentially act as a namespace for any nested types. In the prior example, we were able to access enumerator APPLE directly, because the APPLE enumerator was placed into the global scope (we could have prevented this by using an enum class instead of an enum, in which case we'd have accessed APPLE via FruitType::APPLE instead). Now, because FruitType is considered to be part of the class, we access the APPLE enumerator by prefixing it with the class name: Fruit::APPLE.

Note that because enum classes also act like namespaces, if we'd nested FruitType inside Fruit as an enum class instead of an enum, we'd access the APPLE enumerator via Fruit::FruitType::APPLE.

Other types can be nested too

Although enumerations are probably the most common type that is nested inside a class, C++ will let you define other types within a class, such as typedefs, type aliases, and even other classes!

Like any normal member of a class, nested classes have the same access to members of the enclosing class that the enclosing class does. However, the nested class does not have any special access to the "this" pointer of the enclosing class.

Defining nested classes isn't very common, but the C++ standard library does do so in some cases, such as with iterator classes. We'll cover iterators in a future lesson.

 **8.16 -- Timing your code**

 **Index**