12.17 — Nested types in classes

ALEX AUGUST 2, 2021

Consider the following short program:

```
#include <iostream>

enum class FruitType
{
 apple,
 banana,
 cherry
};

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

public:


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

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

int main()
{
 Fruit apple { FruitType::apple };

 if (apple.getType() == FruitType::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

Much like functions and data can be members of a class, in C++, types can also 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() const { return m_type;  }
 int getPercentageEaten() const { 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, much as enum classes do. In the prior example, because we used an enum class, we had to qualify our enumerators with the FruitType:: scope qualifier. In this example, because FruitType is a normal enum that is part of the class, we access our enumerators using the Fruit:: scope qualifier.

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 enumeration via a Fruit::FruitType:: scope qualifier. This double-scoping is unnecessary, so we've used a normal enum.

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.

One other limitation of nested types -- they can’t be forward declared. However, this is rarely a problem in practice since the entire class definition (including the nested type) can generally be #included where needed.

Defining nested classes isn’t very common, but the C++ standard library does do so in some cases, such as with iterator classes.

Next lesson
12.18 Timing your code

Back to table of contents

Previous lesson
12.16 Anonymous objects

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ coc

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

DP N N FOUT

Newest