13.3 — Overloading operators using normal functions

ALEX AUGUST 8, 2021

In the previous lesson, we overloaded operator+ as a friend function:

```
#include <iostream>

class Cents
{
private:
  int m_cents{};

public:
  Cents(int cents)
    : m_cents{ cents }
  {}

  // add Cents + Cents using a friend function
  friend Cents operator+(const Cents& c1, const Cents& c2);

  int getCents() const { return m_cents; }
};

// note: this function is not a member function!
Cents operator+(const Cents& c1, const Cents& c2)
{
  // use the Cents constructor and operator+(int, int)
  // we can access m_cents directly because this is a friend function
  return { c1.m_cents + c2.m_cents };
}

int main()
{
  Cents cents1{ 6 };
  Cents cents2{ 8 };
  Cents centsSum{ cents1 + cents2 };
  std::cout << "I have " << centsSum.getCents() << " cents.\n";

  return 0;
}
```

Using a friend function to overload an operator is convenient because it gives you direct access to the internal members of the classes you're operating on. In the initial Cents example above, our friend function version of operator+ accessed member variable m_cents directly.

However, if you don't need that access, you can write your overloaded operators as normal functions. Note that the Cents class above contains an access function (getCents()) that allows us to get at m_cents without having to have direct access to private members. Because of this, we can write our overloaded operator+ as a non-friend:

```
#include <iostream>

class Cents
{
private:
  int m_cents{};

public:
  Cents(int cents)
    : m_cents{ cents }
  {}

  int getCents() const { return m_cents; }
};

// note: this function is not a member function nor a friend function!
Cents operator+(const Cents& c1, const Cents& c2)
{
  // use the Cents constructor and operator+(int, int)
  // we don't need direct access to private members here
  return Cents{ c1.getCents() + c2.getCents() };
}

int main()
{
  Cents cents1{ 6 };
  Cents cents2{ 8 };
  Cents centsSum{ cents1 + cents2 };
  std::cout << "I have " << centsSum.getCents() << " cents.\n";

  return 0;
}
```

Because the normal and friend functions work almost identically (they just have different levels of access to private members), we generally won't differentiate them. The one difference is that the friend function declaration inside the class serves as a prototype as well. With the normal function version, you'll have to provide your own function prototype.

Cents.h:

```
#ifndef CENTS_H
#define CENTS_H

class Cents
{
private:
  int m_cents{};

public:
  Cents(int cents)
    : m_cents{ cents }
  {}

  int getCents() const { return m_cents; }
};

// Need to explicitly provide prototype for operator+ so uses of operator+ in other files know this overload exists
Cents operator+(const Cents& c1, const Cents& c2);

#endif
```

Cents.cpp:

```
#include "Cents.h"

// note: this function is not a member function nor a friend function!
Cents operator+(const Cents& c1, const Cents& c2)
{
  // use the Cents constructor and operator+(int, int)
  // we don't need direct access to private members here
  return { c1.getCents() + c2.getCents() };
}
```

main.cpp:

```
#include "Cents.h"
#include <iostream>

int main()
{
  Cents cents1{ 6 };
  Cents cents2{ 8 };
  Cents centsSum{ cents1 + cents2 }; // without the prototype in Cents.h, this would fail to compile
  std::cout << "I have " << centsSum.getCents() << " cents.\n";

  return 0;
}
```

In general, a normal function should be preferred over a friend function if it's possible to do so with the existing member functions available (the less functions touching your classes's internals, the better). However, don't add additional access functions just to overload an operator as a normal function instead of a friend function!

Best practice

Prefer overloading operators as normal functions instead of friends if it's possible to do so without adding additional functions.

Next lesson

13.4 Overloading the I/O operators

Back to table of contents

Previous lesson

13.2 Overloading the arithmetic operators using friend functions

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