# It’s great that you provide operator overloads, but it’s also nice to have names

devblogs.microsoft.com/oldnewthing/20230605-00

June 5, 2023

Raymond Chen

Operator overloading.

Looks great. Reduces verbosity.

Until it doesn’t.

Consider this overloaded function call operator:

```
struct StorageLoader
{
    template<typename DataType>
    DataType operator()(StorageOptions<DataType> const* options);
};
```

The idea is that you can use the function call operator on a `StorageLoader` object to load data from storage, using a `StorageOptions` to describe how you want it to be loaded.

```
StorageOptions<Data1> data1Options;
data1Options.ignore_missing(true);

StorageLoader storageLoader;
Data1 data1 = storageLoader(&data1Options);
```

The parameter is accepted as a pointer so you can pass `nullptr` to indicate that you accept all defaults.

```
// Oops, this doesn't work.
Data1 data1 = storageLoader(nullptr);
```

The `nullptr` doesn’t work because the compiler can’t read your mind to figure out which overload you’re trying to call. You have to help it along, either by refining the type of the parameter:

```
Data1 data1 = storageLoader(static_cast<StorageOptions<Data1>*>(nullptr));
```

or by explicitly specializing the function call operator.

```
Data1 data1 = storageLoader.operator()<Data1>(nullptr);
```

Neither of these is very attractive, and they certainly defeat any conciseness benefit of an overloaded operator.

I personally am a fan of giving named function equivalents to overloaded operators, particularly if they are templated. In this case, I would have done something like

```
struct StorageLoader
{
    template<typename DataType>
    DataType Load(StorageOptions<DataType> const* options);

    template<typename DataType>
    DataType operator()(StorageOptions<DataType> const* options)
    { return Load(options); }
};
```

The function call operator is just a convenient shorthand for calling the `Load` method.

```
// Using function call operator
data1 = storageLoader(&data1Options);

// Using named method
data1 = storageLoader.Load(&data1Options);

// Named method works better for nullptr
data1 = storageLoader.Load<Data1>(nullptr);
```

And then I can make the parameter to `Load` default to `nullptr`:

```
struct StorageLoader
{
    template<typename DataType>
    DataType Load(StorageOptions<DataType> const* options
                  = nullptr);

    ...
};
```

which allows you to write

```
// If no parameters, then use default options
data1 = storageLoader.Load<Data1>();
```

Sometimes, the meaning of an overloaded operator is unclear, in which case having an explicit name also helps avoid confusion over what it does. (I'm looking at you, overloaded address-of operator.)

Also, giving a name to the overloaded operator makes generating a pointer-to-method a little less awkward.