Overloading class member access (C++ only)

Last Updated: 2021-03-05

You overload `operator->` with a nonstatic member function that has no parameters. The following example demonstrates how the compiler interprets overloaded class member access operators:

```
struct Y {
  void f() { };
};

struct X {
 Y* ptr;
 Y* operator->() {
   return ptr;
 };
};

int main() {
  X x;
  x->f();
}
```

The statement `x->f()` is interpreted as `(x.operator->())->f()`.
The `operator->` is used (often in conjunction with the pointer-dereference operator) to implement "smart pointers." These pointers are objects that behave like normal pointers except they perform other tasks when you access an object through them, such as automatic object deletion (either when the pointer is destroyed, or the pointer is used to point to another object), or reference counting (counting the number of smart pointers that point to the same object, then automatically deleting the object when that count reaches zero).

One example of a smart pointer is included in the C++ Standard Library called `auto_ptr`. You can find it in the `<memory>` header. The `auto_ptr` class implements automatic object deletion.

Parent topic:

→ Overloading operators (C++ only)

Related reference:

→ Arrow operator ->