The Destructor and the Assignment Operator

Lecture 7
Section 6.3

Robb T. Koether

Hampden-Sydney College

Fri, Jan 30, 2009
The Destructor

The destructor destroys an object, i.e., it deallocates the memory used by the object.

The destructor is not invoked explicitly.

Type::~Type(); // Prototype;
The destructor is used to destroy an object when it passes out of scope.

A local variable passes out of scope when execution returns from a function.

A variable declared within a block `{ }` passes out of scope when execution leaves that block.

A volatile object passes out of scope when the evaluation of the expression in which it occurs is completed.
Purposes of the Default Constructor

The Destructor

```c
int main()
{
    Vectr v(5, 123);
    {
        Vectr u = 5*v;
    }
    return 0;
}
```

- How many vectors get destroyed and exactly when?
The Automatic Destructor

- The automatic destructor
  - Invokes each data member’s destructor.
  - Deallocates the memory used by the data members.
- The automatic destructor does not deallocate memory that the data members point to.
The **this** Pointer

- Every (non-static) member function has a hidden parameter named **this**.
- **this** is always the first parameter in such a function.
- **this** is a constant pointer to the object `(Type* const this)` that invoked the member function.
- **this** provides us with a name for the invoking object.
When we write the prototype of a member function as

**Apparent Prototype**

\[ \text{Type::func}(\text{params}); \]

the actual prototype is

**Actual Prototype**

\[ \text{Type::func}(\text{Type} \ast \text{const this}, \text{params}); \]

*this* is a constant pointer to an object.
Furthermore, when we write the prototype of a member function as

**Apparent Prototype**

```
Type::func(params) const;
```

the actual prototype is

**Actual Prototype**

```
Type::func(Type const* const this, params);
```

*this* is a constant pointer to a constant object.
Usage of the `this` Pointer

- Inside a member function, we refer to a data member as `member`.
- This is interpreted as `this->member`.
Usage of the `this` Pointer

- Inside a member function, we invoke another member function of the same class by writing `func(params)`.
- This is interpreted as `this->func(params)`.
The assignment operator assigns to an existing object the value of another existing object of the same type.

The assignment operator must be a member function.
Purpose of the Assignment Operator

- The assignment operator is used to assign the value of one object to another object in an assignment statement.
The Assignment Operator

\[\text{Type} & \quad \text{Type::operator} = (\text{const Type} & \text{value}) \]

\[
\begin{align*}
\text{if} \ (\text{this} \neq \ & \text{&value}) \\
\{ \\
// \quad \text{Clear out the old value} \\
// \quad \text{Assign the new value} \\
\} \\
\text{return} \ \ast\text{this};
\end{align*}
\]
The Destructor and the Assignment Operator

Robb T. Koether

The Destructor

The this Pointer

The Assignment Operator

Assignment

Form of the Function `operator=()`

The makeEmpty() and makeCopy() Functions

- `void makeEmpty();`
  
  Clears out the old value of the object.

- `void makeCopy(const Type& value);`
  
  Assign the new value to the object.

- It is convenient write these two member functions and then use them in the copy constructor, the destructor, and the assignment operator.
The Copy Constructor

```cpp
Type::Type(const Type& value) {
    makeCopy(value);
    return;
}
```
The Destructor

Type::~Type()
{
    makeEmpty();
    return;
}
The Assignment Operator

```cpp
Type& Type::operator=(const Type& value) {
    if (this != &value) {
        makeEmpty();
        makeCopy(value);
    }
    return *this;
}
```
The automatic assignment operator uses each data member’s assignment operator to assign values to them from the other object.
Multiple Assignments

- The assignment operator is *right-associative*.
- The statement
  
  \[
  a = b = c = d;
  \]

  is equivalent to
  
  \[
  a = (b = (c = d));
  \]
Multiple Assignments

What about the statements

\[ ((a = b) = c) = d; \]

and

\[ (a = b) = (c = d); \]
Example (The `Vectr` Class)

- Download `vectr.h`.
- Download `vectr.cpp`.
- Download and run `VectrTest.cpp`.
Assignment

Homework

- Read Section 6.3, pages 275 - 286.