C++: Constructors and assignment

Miro Jurišić
meeroh@meeroh.org

Constructors

- Create a new object
- Default constructor

```
Class ();

Class c;
Class* c = new Class ();
```

- Copy constructor

```
Class (const Class& inOriginal);

Class c1 (c2);
Class c1* = new Class (c2)
```

- Other constructors, other copy constructors

Assignment operator

- Assign one object to another

```
Class& operator= (const Class& inOriginal)

c1 = c2;
Class c1 = c2;
```

- Other assignment operators

Compiler-generated constructors

- Default constructor

```
Class::Class ():
    Base (),
    mMember ()
{
}
```

- Copy constructor

```
Class::Class (
    const Class& inOriginal):
    Base (inOriginal),
    mMember (inOriginal.mMember)
{
}
```

Compiler-generated assignment

- Assignment operator

```
Class&
Class::operator = (
    const Class& inOriginal)
{
    Base::operator = (inOriginal);
    mMember = inOriginal.mMember;
    return *this;
}
```

Why write your own constructors and assignment operators

- Resource ownership
- Copy-on-write

Constructors

- Compiler must construct all bases and members before entering constructor

```
class Class {
    private:
        string mString;

    public:
        Class (
            string inString)
            // mString ()
        {
            mString = inString;
        }
};
```

Constructors

- Constructor-initializer is never more expensive than assignment – don't pay the multiple construction cost

```
class Class {
    private:
        string mString;

    public:
        Class (
            string inString):
            mString (inString)
        {
        }
};
```

- Worry about exception safety

Order of construction

- Compiler must reorder initialization order to match declaration order

```
class Class {
    public:
        Class (int inValue);

    private:
        int mFirst;
        int mSecond;
};

Class::Class (
    int inValue):
    mSecond (inValue),
    mFirst (mSecond + 1) // bug
{
}
```

- Likewise for base classes
- gcc 2.9x and 3.x warn, MW C++ compiler in CWP8 and older doesn't

Implicit constructors

- Items not explicitly constructed are always constructed implicitly – this may not be what you want:

```
class Base {
    private:
        int mValue;

    public:
        Base ():
            mValue (0)
        {
        }

        Base (
            int inValue):
            mValue (inValue)
        {
        }
};
```

```
class Derived:
    public Base {
    private:
        mName;

    public:
        Derived ()
            // Base ()
            // mName ()
        {
        }

        Derived (
            string inName):
            // Base ()
            mName (inName)
        {
        }

        Derived (
            const Derived& inOriginal):
            // Base () -- bug?
            mName (inOriginal.mName)
        {
        }
};
```

Assignment operator return value

```
a = b = c
a.operator = (b.operator = (c))
```

- Return *this from assignment operator to allow assignment chains

```
class Class
{
    private:
        string mName;

    public:
        Class&
        operator = (
            const Class& inOriginal)
        {
            mName = inOriginal.mName;
            return *this;
        }
};
```

Assignment to self

```
a = a;

Class a;
// lots of code
Class* b = &a;
// lots of code
*b = a;
```

- Check for assignment to self in assignment operator

```
class Class
{
    private:
        int* mValue;

    public:
        Class (
            int inValue):
            mValue (new int (inValue))
        {
        }
```

```
~Class ()
{
    delete (mValue);
}

Class (
    const Class& inOriginal):
    mValue (new int (inOriginal.mValue))
{
}

Class&
operator = (
    const Class& inOriginal)
{
    delete mValue;
    mValue = new int (inOriginal.mValue); // bug
    return *this;
}
```

```
        Class&
        operator = (
            const Class& inOriginal)
        {
            if (&inOriginal != this) {
                delete mValue;
                mValue = new int (inOriginal.mValue);
            }

            return *this;
        }
};
```

- Be aware of object identity and object equality

Base assignment

- Copy everything, including bases, in assignment operator

```
class Base {
    private:
        int mValue;

    public:
        Base ():
            mValue (0)
        {
        }

        Base (
            int inValue):
            mValue (inValue)
        {
        }
};
```

```
class Derived:
    public Base {
    private:
        mName;

    public:
        Derived&
        operator = (
            const Derived& inOriginal)
        {
            if (&inOriginal != this) {
                Base::operator = (inOriginal);
                mName = inOriginal.mName;
            }

            return *this;
        }
};
```

Preventing copying

- Do not omit copy constructor and assignment operator for non-copyable classes
- Prevent copying with unimplemented private copy constructor and assignment operator

```
class NonCopyable {
    private:
        NonCopyable (const NonCopyable&); // Omit implementation
        NonCopyable& operator= (const NonCopyable&); // Omit implementation
};
```