8.11 Rule of three

Classes have three special member functions that are commonly implemented together:

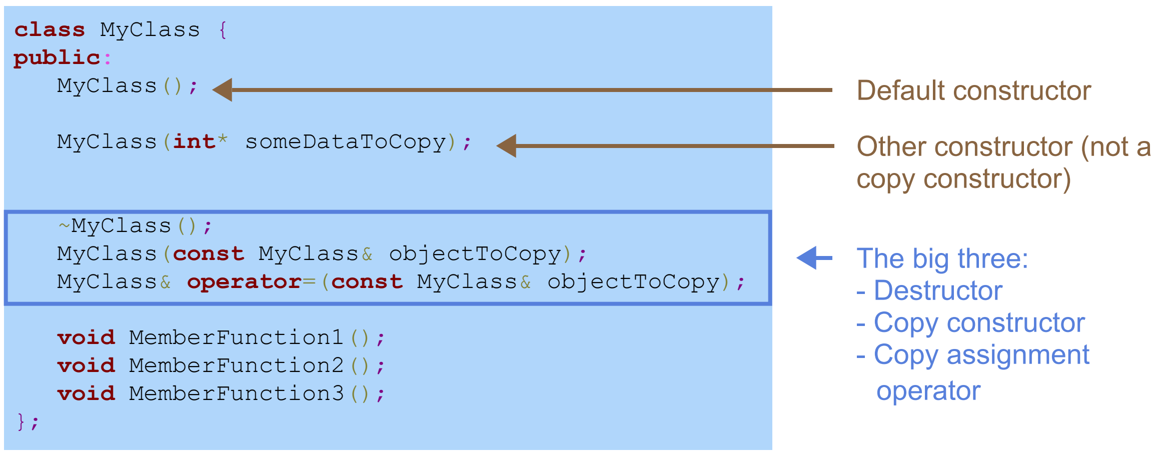
- **Destructor:** A destructor is a class member function that is automatically called when an object of the class is destroyed, such as when the object goes out of scope or is explicitly destroyed as in `delete someObject;`.
- **Copy constructor:** A copy constructor is another version of a constructor that can be called with a single pass by reference argument. The copy constructor is automatically called when an object is passed by value to a function, such as for the function `SomeFunction(MyClass localObject)` and the call `SomeFunction(anotherObject)`, when an object is initialized when declared such as `MyClass classObject1 = classObject2;`, or when an object is initialized when allocated via "new" as in `newObjectPtr = new MyClass(classObject2);`
- **Copy assignment operator:** The assignment operator "=" can be overloaded for a class via a member function, known as the copy assignment operator, that overloads the built-in function "operator=", the member function having a reference parameter of the class type and returning a reference to the class type.

The ***rule of three*** describes a practice that if a programmer explicitly defines any one of those three special member functions (destructor, copy constructor, copy assignment operator), then the programmer should explicitly define all three. For this reason, those three special member functions are sometimes called ***the big three***.

A good practice is to always follow the rule of three and define the big three if any one of these functions are defined.

PARTICIPATION ACTIVITY 8.11.1: Rule of three.

1 2 3 2x speed


Default constructor
Other constructor (not a copy constructor)
The big three:
- Destructor
- Copy constructor
- Copy assignment operator

```
class MyClass {
public:
   MyClass();

   MyClass(int* someDataToCopy);


   ~MyClass();
   MyClass(const MyClass& objectToCopy);
   MyClass& operator=(const MyClass& objectToCopy);

   void MemberFunction1();
   void MemberFunction2();
   void MemberFunction3();
};
```

A constructor may exist that copies some data, but isn't the copy constructor. The copy constructor for MyClass takes a const MyClass& argument.

Feedback?

Default destructors, copy constructors, and assignment operators

- *If the programmer doesn't define a destructor for a class, the compiler implicitly defines one having no statements, meaning the destructor does nothing.*
- *If the programmer doesn't define a copy constructor for a class, then the compiler implicitly defines one whose statements do a memberwise copy, i.e., `classObject2.memberVal1 = classObject1.memberVal1`, `classObject2.memberVal2 = classObject1.memberVal2`, etc.*
- *If the programmer doesn't define a copy assignment operator, the compiler implicitly defines one that does a memberwise copy.*

PARTICIPATION ACTIVITY 8.11.2: Rule of three. ✓

1) If the programmer does not explicitly define a copy constructor for a class, copying objects of that class will not be possible. ✓

True [✓ False]

Correct

If the programmer doesn't define a copy constructor for a class, then the compiler automatically defines one that performs a memberwise copy.

2) The big three member functions for classes include a destructor, copy constructor, and default constructor. ✓

True [✓ False]

Correct

The big three member functions for classes include a destructor, copy constructor, and copy assignment operator.

3) If a programmer explicitly ✓

defines a destructor, copy constructor, or copy assignment operator, it is a good practice to define all three.

[✓ True] False

Correct

A good practice, known as the rule of three, is if a programmer explicitly defines any one of those three special member functions (destructor, copy constructor, copy assignment operator), then the programmer should probably explicitly define all three.

4) Assuming `MyClass prevObject` has already been declared, the statement `MyClass object2 = prevObject;` will call the copy assignment operator.

True [✓ False]

Correct

A class's copy constructor, not copy assignment operator, will be called when an object is initialized by copying another object during declaration.

5) Assuming `MyClass prevObject` has already been declared, the following variable declaration will call the copy assignment operator.

```
MyClass object2;
...
object2 = prevObject;
```

[✓ True] False

Correct

An assignment from one existing object to another existing object will call the copy assignment operator.

Feedback?

Exploring further:

- More on Rule of Three in C++ from GeeksforGeeks.