For example, consider struct `S4` in the code snippet below in which we have chosen to make explicit that the copy operations are to be autogenerated by the compiler~~; note, in particular, that implicit declaration and generation of each of the other special member functions are left unaffected.~~

```
struct S4
{
    S4(const S4&) = default;             // copy constructor
    S4& operator=(const S4&) = default;  // copy-assignment operator

    ~~// has no effect on other four special member functions, i.e.,~~
    ~~// implicitly generates the default constructor, the destructor,~~
    ~~// the move constructor, and the move-assignment operator~~
};
```

A defaulted declaration may appear with any **access specifier** (i.e., **`private`**, **`protected`**, or **`public`**), and access to that generated function will be regulated accordingly:

```
struct S5
{
private:
    S5(const S5&) = default;             // private copy constructor
    S5& operator=(const S5&) = default;  // private copy-assignment operator

protected:
    ~S5() = default;                     // protected destructor

public:
     S5() = default;                     // public default constructor
};
```

In the example above, copy operations exist for use by *member* and *friend* functions only. Declaring the destructor **`protected`** or **`private`** limits which functions can create automatic variables of the specified type to those functions with the appropriately privileged access to the class. Declaring the default constructor ~~**`public`**~~ is necessary to avoid its declaration being suppressed by another constructor — e.g., the private copy constructor in the code snippet above — or *any* move operation.

In short, using `=`**`default`** on the first declaration denotes that a special member function is intended to be generated by the compiler, irrespective of any user-provided declarations; in conjunction with `=`**`delete`** (see Section 1.1."Deleted Functions" on page 53), using `=`**`default`** affords the fine-grained control over which special member functions are to be generated and/or made publicly available.

Defaulting the implementation of a user-provided special member function

The `=`**`default`** syntax can also be used after the first declaration but with a distinctly different meaning: The compiler will treat the first declaration as a user-provided special