Deleted Functions

Consider a class, `FileHandle`, that uses the **RAII** idiom to safely acquire and release an I/O stream. As **copy semantics** are typically not meaningful for such resources, we will want to suppress generation of both the **copy constructor** and **copy assignment operator**. Prior to C++11, there was no direct way to express suppression of special member functions in C++. The commonly recommended workaround was to **declare** the two methods **private** and leave them unimplemented, typically resulting in a compile-time or link-time error when accessed:

```
#include <cstdio>  // FILE
class FileHandle
{
private:
    // ...

    FileHandle(const FileHandle&);             // not implemented
    FileHandle& operator=(const FileHandle&);  // not implemented

public:
    explicit FileHandle(FILE* filePtr);
    ~FileHandle();

    // ...
};
```

Not implementing a special member function that is declared to be private ensures that there will be at least a link-time error in case that function is inadvertently accessed from within the implementation of the class itself. With the **= delete** syntax, we are able to (1) explicitly express our intention to make these special member functions unavailable, (2) do so directly in the **public** region of the class, ~~and~~ (3) enable clearer compiler diagnostics:

```
class FileHandle
{
private:
    // ...
    // Declarations of copy constructor and copy assignment are now public.

public:
    explicit FileHandle(FILE* filePtr);
    ~FileHandle();

    FileHandle(const FileHandle&) = delete;             // make unavailable
    FileHandle& operator=(const FileHandle&) = delete;  // make unavailable

    // ...
};
```