```
String s;  // No memory is allocated.
```

To avoid having to check, internally, on each access whether a given string representation is a **null pointer value**, we instead create a common **`static`** *empty* string, `s_empty`, nested within our `String` class, and install its address during **default construction** or when we would otherwise want to represent an empty string. This address serves as a **sentinel** whose requisite runtime checking is properly relegated to more costly and/or presumably less frequent operations, such as copy construction, assignment, and destruction. Thus, an added **object invariant** is that a string **value** whose representation is dynamically allocated is never empty.

Finally, to provide better factoring, the **definition** of our **value-semantic** `String` class **declares** a private **`static`** **member function**, `dupStr`, that dynamically allocates and populates a new block of memory exactly sized to hold a supplied, nonempty **null-terminated-string** value:

```
// my_string.h:
// ...

class String  // greatly simplified null-terminated-string manager
{
    const char* d_str_p;  // immutable value, often allocated dynamically

    static const char s_empty[1];  // empty, used as sentinel indicating null

    static const char* dupStr(const char* str);  // allocate/return copy of str

public:
    // C++03
    String();                             // default constructor
    String(const char* value);            // value constructor
    String(const String& original);       // copy constructor
    ~String();                            // destructor
    String& operator=(const String& rhs);  // copy-assignment operator
    const char* str() const;

    /* C++11 (to be added later)
    String(String&& expiring) noexcept;    // move constructor
    String& operator=(String&& expiring);  // move-assignment operator
    */
};
```

Perhaps the best way to unpack the C++03 class **definition** of `String` in the code snippet above is to **define** its **members** in order of **declaration** in a corresponding `.cpp` file, realizing, of course, that all but the **definition** of the **`static`** **data member**, `s_empty`, would most likely be moved to the `.h` file as **`inline`** functions: