

Listing 5

A C++ String Class	§1
Representing an Xstring	§4
Construction and Destruction	§5
Assignment	§12
Miscellaneous Operations	§14
References	§18
Index	§19

Copyright © 1994 by Lee Wittenberg

Portions copyright © 1991 by AT&T Bell Telephone Laboratories, Inc.

1. A C++ String Class. To demonstrate the use of **CWEB** for C++ programming, we adapt the **string** class described by Stroustrup [1, pages 248–251]. Explanations in *slanted type* (including inline comments, when possible) are direct quotes from the original. We make a few minor changes along the way, but on the whole, we stick to Stroustrup’s design.

2. We put the interface part of our class in the header file **xstring.h**. We call our class “**Xstring**” rather than “**string**” to avoid confusion with the original and other (more useful) string classes. We restrict ourselves to a lowercase file name to maintain portability among operating systems with case-insensitive file names.

```
<xstring.h 2> ≡
#ifndef XSTRING_H
#define XSTRING_H    // prevent multiple inclusions
    class Xstring {
        <Private Xstring members 4>
    public:
        <Public Xstring members 5>
    };
#endif
```

This code is cited in section 3.

This code is used in section 3.

3. We implement the class members in a single “unnamed chunk” that will be tangled to **xstring.c** (or **xstring.cc** or **xstring.cpp**, depending on your compiler’s preference). We include the contents of **<xstring.h 2>** directly, rather than relying on **#include**, because we can.

```
<Header files 8>
<xstring.h 2>
<Xstring members and friends 6>
```

Listing 5 (continued)

4. Representing an Xstring. The internal representation of an **Xstring** is simple. *It counts the references to a string to minimize copying and uses standard C++ character strings as constants.*

```
{ Private Xstring members 4 } ≡  
  struct srep {  
    char *s;    // pointer to data  
    int n;      // reference count  
    srep() { n ← 1; }  
  };  
  srep *p;
```

See also section 16.

This code is used in section 2.

5. Construction and Destruction. *The constructors and the destructor are trivial.* We use the null string as a default constructor argument rather than a null pointer to protect against possible **string.h** function anomalies.

```
{ Public Xstring members 5 } ≡  
  Xstring(const char *s ← "");    // Xstring x ← "abc"  
  Xstring(const Xstring &);    // Xstring x ← Xstring ...  
  ~Xstring();
```

See also sections 12, 14, and 15.

This code is used in section 2.

6. An **Xstring** constructed from a standard string needs space to hold the characters:

```
{ Xstring members and friends 6 } ≡  
  Xstring::Xstring(const char *s)  
  {  
    p ← new srep;  
    { Allocate space for the string and put a copy of s there 7 };  
  }
```

See also sections 9, 10, 13, and 17.

This code is used in section 3.

7. There is always the possibility that a client will try something like “**Xstring** x ← Λ.” We substitute the null string whenever we are given a null pointer.

```
{ Allocate space for the string and put a copy of s there 7 } ≡  
  if (s ≡ Λ) s ← "";  
  p→s ← new char [strlen(s) + 1];  
  strcpy(p→s, s);
```

This code is used in sections 6 and 13.

8. { Header files 8 } ≡
#include <string.h> // Standard C header for *strcpy*

This code is used in section 3.

9. On the other hand, to build an **Xstring** from another **Xstring**, we only have to increment the reference count:

Listing 5 (continued)

```

⟨ Xstring members and friends 6 ⟩ +≡
Xstring::Xstring(const Xstring &x)
{
    x.p-n++;
    p ← x.p;
}

```

10. The destructor also has to worry about the reference count:

```

⟨ Xstring members and friends 6 ⟩ +≡
Xstring::~Xstring()
{
    ⟨ Decrement reference count, and remove p if necessary 11 ⟩;
}

```

```

11. ⟨ Decrement reference count, and remove p if necessary 11 ⟩ ≡
if (--p-n ≡ 0) {
    delete []p-s;
    delete p;
}

```

This code is used in sections 10 and 13.

12. Assignment. *As usual, the assignment operators are similar to the constructors. They must handle cleanup of their first (left-hand) operand:*

```

⟨ Public Xstring members 5 ⟩ +≡
Xstring &operator←(const char *);
Xstring &operator←(const Xstring &);

```

```

13. ⟨ Xstring members and friends 6 ⟩ +≡
Xstring &Xstring::operator←(const char *s)
{
    if (p-n > 1) {    // disconnect self
        p-n--;
        p ← new srep;
    } else    // free old string
        delete []p-s;
    ⟨ Allocate space for the string and put a copy of s there 7 ⟩;
    return *this;
}

Xstring &Xstring::operator←(const Xstring &x)
{
    x.p-n++;    // protect against “st ← st”
    ⟨ Decrement reference count, and remove p if necessary 11 ⟩;
    p ← x.p;
    return *this;
}

```

Listing 5 (continued)

14. Miscellaneous Operations. We provide a conversion operator to translate **Xstring**'s into ordinary strings. This allows us to pass them to standard functions like *strlen* (and gives us an output operator for free). We convert to **const** strings to prevent strange things from happening if a client should try to use a standard function like *strcat* to modify an **Xstring**.

```
⟨ Public Xstring members 5 ⟩ +≡  
    operator const char *() { return p-s; }
```

15. *The subscript operator is provided for access to individual characters. The index is checked.* However, we depart from the original design by returning a dummy element when the index is out of bounds rather than generating an error message (or an exception).

```
⟨ Public Xstring members 5 ⟩ +≡  
    char &operator [] (int i) {  
        return ((i < 0) ∨ (strlen(p-s) < i) ? dummy : p-s[i]); }
```

16. ⟨ Private **Xstring** members 4 ⟩ +≡
 static char dummy;

17. ⟨ **Xstring** members and friends 6 ⟩ +≡
 char **Xstring**::dummy;

18. References.

[1] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, second edition, 1991.

19. Index.

<i>dummy</i> : 15, <u>16</u> , 17.	<i>strcat</i> : 14.
<i>i</i> : <u>15</u> .	<i>strcpy</i> : 7, 8.
<i>n</i> : <u>4</u> .	<i>strlen</i> : 7, 14, 15.
operator : <u>12</u> , <u>13</u> , <u>14</u> , <u>15</u> .	<i>x</i> : <u>5</u> , <u>7</u> , <u>9</u> , <u>13</u> .
<i>p</i> : <u>4</u> .	Xstring : <u>2</u> , <u>6</u> , <u>9</u> , <u>10</u> , <u>13</u> .
<i>s</i> : <u>4</u> , <u>5</u> , <u>6</u> , <u>13</u> .	XSTRING_H : <u>2</u> .
srep : <u>4</u> , 6, 13.	

⟨ Allocate space for the string and put a copy of *s* there 7 ⟩ Used in sections 6 and 13.

⟨ Decrement reference count, and remove *p* if necessary 11 ⟩ Used in sections 10 and 13.

⟨ Header files 8 ⟩ Used in section 3.

⟨ Private **Xstring** members 4, 16 ⟩ Used in section 2.

⟨ Public **Xstring** members 5, 12, 14, 15 ⟩ Used in section 2.

⟨ **xstring.h** 2 ⟩ Cited in section 3. Used in section 3.

⟨ **Xstring** members and friends 6, 9, 10, 13, 17 ⟩ Used in section 3.